MICRO™

The Magazine of the APPLE, KIM, PET and Other 6502 Systems

APPLE II HIRES

Picture Compression

NO. 18 NOVEMBER 1979 $2.00

SUPERKIM

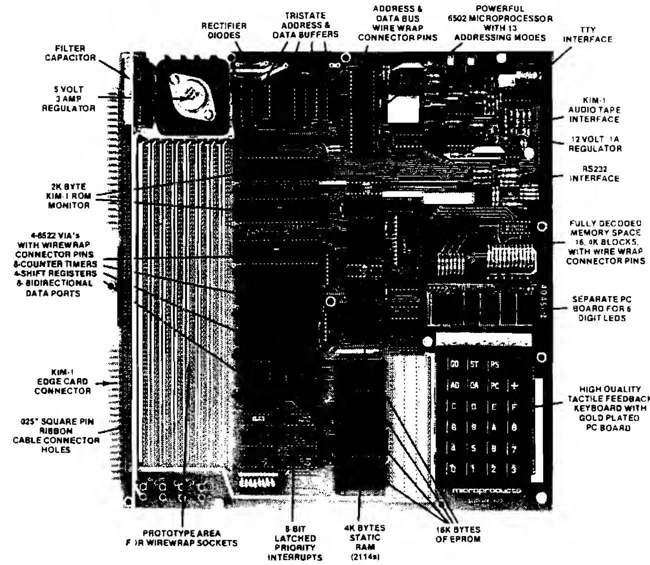
INDUSTRIAL CONTROL COMPUTER

Here is a powerful microprocessor control system development tool and a complete real-time multitasking microcomputer in one package. There is no need to buy a power supply, motherboard, memory boards and separate I/O boards when your requirements may be satisfied by a **SUPER-KIM.** You may only need a couple of wire-wrap sockets and a few LSI chips installed in the big 3" x 10" onboard prototype area to accomplish the required memory expansion and interface with the real world.

Some single chip interface devices available are: UARTS, 16 channel-8 bit analog to digital data acquisition systems, floppy disk controllers and dot matrix printer controllers. Furthermore, you will shortly be able to buy single 5 volt supply pseudo static 8K byte (that's right, you read it right, 8K x 8 bits) memory chips in a single 28 pin package. These chips use the same technology developed for the 64K bit dynamic RAMs now being manufactured by TI, MOTOROLA and others. Just five of these chips and four 2732 EPROMs in the sockets already supplied in the **SUPERKIM** will yield a fully populated **SUPERKIM** with 44K bytes of RAM, 16K bytes of EPROM with serial and parallel I/O ports, and enough room leftover in the prototype area for a LSI floppy disk controller chip. Zilog already has, on the market, a 4K byte version of this memory chip that is pin compatible with the 8K byte version; no need to rewire your sockets when the larger memories become available. Put in 24K now and upgrade later to 44K.

If you started with a KIM-1, SYM-1 or AIM-65 and tried to expand it to the basic capabilities of the **SUPERKIM**, you would need a power supply ($60), a motherboard ($120), a prototype board ($30), a memory board ($120), and an I/O board ($120) for a total cost of from $620 in the case of the KIM-1 to $825 in the case of the AIM-65. You still would not have real time multitasking capabilities.

Multitasking is a situation where the microcomputer appears to be doing more than one job simultaneously. For example, the



microcomputer could be sending data to a printer, accepting analog data from a 16-channel data acquisition system and presenting data to an operator monitoring a LCD or LED display, all the while keeping track of time.

Multitasking is accomplished on the SUPER-KIM by use of vectored priority interrupts and a real time clock. This real time clock is implemented using one of the four on-board 6522 programmable tone generators.

The **SUPERKIM**, with its keyboard, display and ROM monitor, can be used as a system analyzer for troubleshooting hardware and software in-the-field or during system development as an in circuit emulator. The monitor can stop the CPU at any point in the program, step through the program, change the contents of the systems' memory and CPU registers, and record the CPU's registers during a selected portion of the program. It offers one of the most powerful combinations of development and diagnostic tools available on the market today.

All of the above is unavailable on any other singleboard computer at any price.

$395.00

microproducts

2107 ARTESIA BOULEVARD • REDONDO BEACH, CA 90278 • (213) 374-1673

MICRO™

Table of Contents

Dual Tape Drive for SYM - 1 BASIC by George Wells	5
Some Useful Memory Locations and Subroutines for OSI BASIC in ROM by S. R. Murphy	9
A Tape Indexing System for the PET by Alan R. Hawthorne	11
Subroutine Parameter Passing by Mark Swanson	14
APPLE II Hires Picture Compression by Bob Bishop	17
Assembly Language Applesoft Renumber by Alan D. Floeter	27
Performing Math Functions in Machine Language by Alfred J. Bruey	30
TSAR: A Time Sharing Administrative Routine for the KIM-1 by Philip K. Hooper	35
Interfacing the CI-812 to the KIM by Jim Dennis	43
MICROBES: Ampersort	45
SYM-1 Baudot TTY Interface by Richard A. Leary	49
The MICRO Software Catalog: XIV by Mike Rowe	55
Alarming APPLE by Paul Irwin	59
6502 Bibliography: Part XIV by William R. Dial	61

November 1979
Issue Number 18

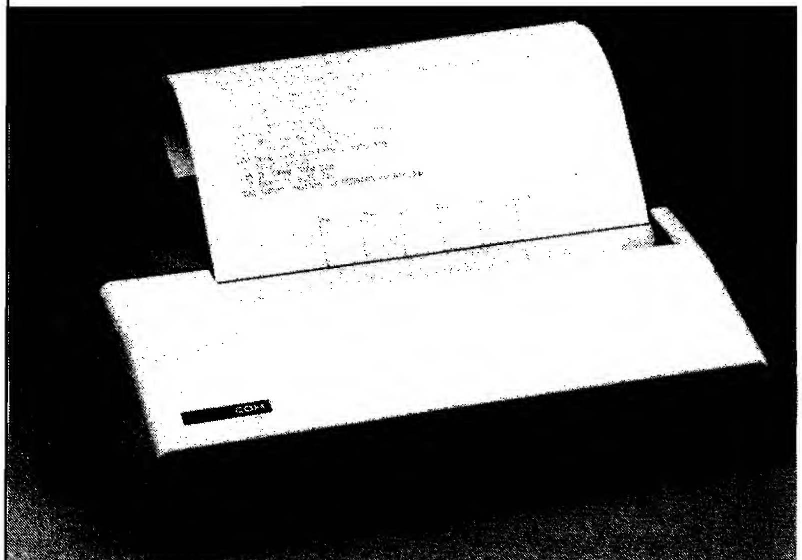
Staff

Advertiser's Index

Beta Computer Devices	31	Microproducts	1
Classified Ads	13,20	Optimal Technology, Inc.	28
COMPAS	2	Powersoft, Inc.	25
Computer World	32,33	Programma International	BC
The Computerist, Inc.	26	Progressive Software	16
Computer Shop	8	Rainbow Computing, Inc.	IBC
Connecticut microComputers	46,47,48	RNB - Enterprises	64
Electronic Specialists, Inc.	7	SKYLES Electronic Works	4,14,15
Enclosures Group	34	Softside Software	48
H. Geller Computer Systems	41	Softouch, Inc.	7
Home Computer Center	IFC	Synergistic Software	60
Hudson Digital Electronics	42	Weldon Electronics	63
MICRO	10,48	West Side Electronics	60
MICRO MUSIC Inc.	57		

Skyles Electric Works

PAL-80™



- 80 characters per line
- 8½ inch wide thermal paper
- Full graphics at 60 dots/inch
- Interfaced to PET
- Works with all PET peripherals
- 40 character per second rate
- Microprocessor controlled
- Bidirectional look-ahead printing
- Quiet operation
- No external power supplies
- Only two driven parts
- High reliability
- Clear 5 x 7 characters
- Attractive metal and plastic case

The Skyles PAL-80™ is a high speed thermal printer offering the combination of text printing at 80 characters per line and continuous graphics at 60 dots per inch. In the text mode, upper and lower case data are printed at 40 characters per second. The 5 x 7 characters provide clear readable copy on white paper; no hard to find, hard to read aluminized paper.

In the graphics mode, seven bits of each byte correspond to the seven dots in each of the 480 print positions per line. Since the computer driving the printer has full control over every print position, it can print graphs, bar charts, line drawings, even special and foreign language symbols. Despite its low cost, the Skyles PAL-80 is a true intelligent printer with full line buffering and bidirectional look-ahead printing.

High reliability is designed in: The thick film thermal print head has a life expectancy of 100,000,000 characters. Two DC stepping motors provide positive control of the print head and the paper drive.

The Skyles PAL-80 operates directly from a 115V 60 Hz line (230V 50 Hz available). No external power supplies are required.

It comes complete with an interface for the PET: a two and a half foot cable plugs into the IEEE interface at the back of your PET. Works with all PET models and PET or Skyles peripherals.

Please send me________ **Skyles PAL-80 printer(s)** complete with 2½ foot interface cable to attach to my PET at **$675.00 each*** (Plus $10.00 shipping and handling). I also will receive a test and graphics demonstration tape at no additional charge and over 150 feet of 8½ inch wide black on white thermal paper $_________

I would also like to order ______ rolls of 8½ inch wide by 85 ft. long thermal paper (black ink) at $5.00 each $ ____________

________ 10 roll cartons at $45.00 $____________

VISA, Mastercharge orders call **(800) 227-8398**
California orders call **(415) 494-1210**

*California residents add 6 to 6½% sales tax where applicable.

PAL-80 SPECIFICATIONS

TEXT	
Format	80 characters per eight inch line 6 lines per inch nominal
Print speed	40 characters per second
Line Feed	50 milliseconds nominal
Character Set	96 Characters, including upper and lower case, numerals, and symbols
GRAPHICS	
Format	480 print positions per line
Print Speed	240 print positions per second
COMMON	
Paper	8½ inch wide thermal paper, available in 85 foot folls, black image on white
Dimension	12"W x 10"D x 2¾"H
Weight	8 lbs (3.6 kg)

Skyles Electric Works | 0301 Stonydale Drive
Cupertino, CA 95014 • (408) 735-7891

Dual Tape Drive for SYM-1 BASIC

If you want to make your SYM - 1 BASIC work with two tape recorders and manage tape cassette files, here is what it takes. A few important observations about the BASIC are presented that could save you grief.

George Wells
1620 Victoria Place
La Verne, CA 91750

When I bought my SYM-1, I had no intention of buying BASIC for it. However, after not being able to show off my new computer to my friends and relatives in a way they could understand, I decided to go ahead and get the BASIC ROMS. Then I purchased a book of BASIC games and copied several of them onto tape. The need to have a convenient means of copying tapes to make backup copies became apparent. Also, I discovered that the tape routines do not work after BASIC has been interrupted and reentered with a "GO" command (warm start). After I recieved the tech note from Synertek describing how to put trig functions in BASIC, I found out how to fix this problem. The tape routines use system RAM to pass information from BASIC and apparently the call to "ACCESS" was omitted during the warm start.

To make the second tape recorder work, I added five components to one of the buffered outputs to make it look like the audio cassette remote control configured for type IV (see figure 3-3, pages 3-7 of SYM reference manual). This is the set up required for one of the recommended recorders, Radio Shack CTR-40. Refer to SYM reference manual figure 4-5A, pages 4-12. Note that pad 1 is located between pads 2 and 6. The following connections were made to buffer PB4:

Install 470 OHM at location R5.

Install 2N2902A transistor emitter to pad 6 base to pad 9 collector to pad 1.
Install 1K resistors between pads 6&9.
Install 1N914 diode anode to pad 1 cathode to pad 6
Install 1N914 diode anode to pad 19 cathode-pad 16.
Install subminiature phono plug tip to pad 6, shield to pad 1.

My first tape recorder (General Electric model M8455A) is connected to the normal remote control configured for type V. The audio out (LO) goes to the MIC input. I discovered a trim pot on the inside of the recorder which, if turned completely counter-clockwise, makes the recording ideal for the SYM computer (terrible for voice though). Also, I found it necessary to align the heads of both tape recorders before I could get reliable operation. The GE recorder is used normally to save files and the Radio Shack recorder is used with special routines to load files. The assembly language program was written at a location just before the trig function routines and includes two sets of execute commands for cold and warm starts to BASIC that are compatible with the trig functions and include a call to "ACCESS" so that the tape routines will work after a warm start.

The hex dump of the tape drive routine plus the trig functions (from Synertek Systems Corp. Tech Note 53) can be used to enter the code into your system. Use the same verify command and compare checksums to check your work. I save this file on tape using an ID of $31 which can be loaded and saved from BASIC as file "1".

The sample run-stream illustrates how to make it all work. First, a cold start to BASIC was performed with the monitor execute command (E E5A). SYM responds with everything down to line 100 which was entered to exercise the trig functions and provide something to save on tape. After running the single line BASIC program, it was saved on tape with the file name "T". "NEW" erases the program to indicate that the tape will do a real load. The "USR" command to hex address 8035 takes us out of BASIC and back to the monitor. To get back to BASIC use the execute command (E E95). The response includes everything down to the word "list". Since nothing was listed, this shows that the previous program has been erased. It is loaded back in by transferring the cassette from the "save only" recorder (in my case the GE) and putting it on the "load only" recorder (Radio Shack) and pushing the rewind button. If this is the first time that the load only recorder has been used since the SYM was reset, then the recorder will start rewinding immediately. Otherwise it will wait until the "LOAD T" command is entered. When the tape is rewound, the play button is pressed and the recorder stops automatically when the file is loaded. Listing and running the program show it to be the same as before.

The way I use routines to manage files is through the use of three identical cassette tapes each storing one copy each of all my BASIC programs. I use a fourth tape for temporary storage of a program I am currently working on. When I want to make a copy of all the programs on tape, I put that tape into the LOAD-ONLY or READ-ONLY recorder, and push the PLAY button. Then I put the tape that I want to copy to in the SAVE-ONLY or WRITE-ONLY recorder and push the PLAY-RECORD buttons. I also keep a directory on paper of the program files ID's on tape. Its a simple matter to type a sequence of BASIC commands consisting of a series of LOAD A, SAVE A, LOAD B, SAVE B, LOAD C, SAVE C, etc. If I want to insert a new program from my temp tape, I just swap tapes in the READ-ONLY recorder to get the new program out, and then swap back to continue with the old programs.

```
.E E5A
.J 0
MEMORY SIZE? 3674
WIDTH? 80
OK
POKE202,169:POKE203,14:POKE196,104:POKE197,15

OK
100 PRINT SIN(1),COS(2),TAN(3),ATN(4)
RUN
 .841470985    -.416146836    -.142546543     1.32581766

OK
SAVE T
SAVED

OK
NEW

OK
?USR(&"8035",0)

CB6D,3
.E E95
.G 0

OK
?USR(&"8B86",0)
 0

OK
LIST

OK
LOAD T
LOADED
OK
LIST

 100 PRINT SIN(1),COS(2),TAN(3),ATN(4)
OK
RUN
 .841470985    -.416146836    -.142546543     1.32581766

OK
```

As a matter of habit I then read the tape I have just written to verify that it is O.K. and use it to copy into my third permanent tape. Then I repeat the process going from the third tape back to the original one. Finally, I read the original tape to verify it. If at any point I detect a bad load, I know that I will always have an available on one of my tapes a copy of the file in good condition, that hasn't been overwritten yet.

Small changes can be made in any program file by copying it onto the temp tape with the changes (I usually make two or three copies on the temp tape) and then rewriting the file on each of the permanent tapes by reading the file immediately before the one I want to change to find where to start, and reloading from the temp tape before actually saving the changed file.

Three Other Observations

1.Two words have been omitted from the list of reserved words on page 9 of the BASIC manual: "GO" and "GET". "GO" allows you to spell "GOTO" as "GO TO" if you want; not really a good idea since it takes three bytes of storage instead of only one. "GET" must be a leftover since it always generates an FC error.

2.Page C-2 of the manual states that 6 bytes of storage are used for each variable: 2 for the name and 4 for the value. In fact, 5 are used for the value, bringing the total to 7. This is what gives SYM BASIC its 9+ digit resolution. The disadvantage is that every simple variable (including integer and string variables which only need two and three bytes respectively for their values) uses more bytes than are usually needed. Incidentally, there is a memory saving when using integer or string arrays. However, Microsoft BASIC converts integer values to floating points before using them, which takes longer than using floating points in the first place. Therefore, as a general rule, integer variables should only be used in arrays, and only when it is necessary to conserve memory space.

3.Don't make a mistake when typing a line that prints a hex-formatted number. If you don't follow the format exactly, BASIC hangs up in a loop, printing zeroes. If this occurs, you can recover by doing a reset and going back to BASIC with a warm start. Your program will still be there, but as with any error, the program cannot be continued.

ASSEMBLY LANGUAGE PROGRAM

```
                     MODE    EQU  $FD
                     CONFIG  EQU  $89A5
                     ZERCK   EQU  $832E
                     P2SCR   EQU  $829C
                     DDR3B   EQU  $AC02
                     OR3B    EQU  $AC00
                     LOADT   EQU  $8C78

                             ADR  $E5A
0E5A  4A 30                  ASCII          'J0'      BASIC COLD START COMMAND
0E5C  0D                     BYTE           $0D       CARRIAGE RETURN
0E5D  33 36 37 34            ASCII          '3674'    MEMORY SIZE
0E61  0D                     BYTE           $0D       CARRIAGE RETURN
0E62  38 30                  ASCII          '80'      LINE WIDTH
0E64  14 0D                  BYTE           $14,$0D   CONTROL T, CARRIAGE RETURN
                                 CHANGE TAPE LOAD VECTOR
0E66  50 4F 4B 45            ASCII          'POKE202,169:POKE203,14:
0E6A  32 30 32 2C
0E6E  31 36 39 3A
0E72  50 4F 4B 45
0E76  32 30 33 2C
0E7A  31 34 3A

                                 CHANGE TRIG VECTOR
0E7D  50 4F 4B 45            ASCII          'POKE196,104:POKE197,15'
0E81  31 39 36 2C
0E85  31 30 34 3A
0E89  50 4F 4B 45
0E8D  31 39 37 2C
0E91  31 35
0E93  0D 00                  BYTE           $0D,$00   CARRIAGE RETURN, END EXECUTE

0E95  47 30                  ASCII          'G0'      BASIC WARM START COMMAND
0E97  0D                     BYTE           $0D       CARRIAGE RETURN
                                 JUMP TO MONITOR ACCESS SUBROUTINE
0E98  3F 55 53 52            ASCII          '?USR(&"8B86",0)'
0E9C  28 26 22 38
0EA0  42 38 36 22
0EA4  2C 30 29
0EA7  0D 00                  BYTE           $0D,$00   CARRIAGE RETURN, END EXECUTE

0EA9  84 FD                  STY  MODE      DO CUSTOM INITIALIZE FOR READ RECORDER
0EAB  A9 09                  LDA  #9
0EAD  20 A5 89               JSR  CONFIG
0EB0  20 2E 83               JSR  ZERCK
0EB3  20 9C 82               JSR  P2SCR
0EB6  A9 10                  LDA  #%00010000
0EB8  8D 02 AC               STA  DDR3B     BIT PB 4 OF VIA 3 SET OUTPUT
0EBB  8D 00 AC               STA  OR3B      TURN ON READ TAPE RECORDER
0EBE  20 7B 8C               JSR  LOADT+3   LOAD TAPE BUT SKIP INITIALIZE
0EC1  A9 00                  LDA  #%00000000
0EC3  8D 00 AC               STA  OR3B      TURN OFF READ TAPE RECORDER
0EC6  60                     RTS
```

HEX DUMP

```
.V E5A-FFF
0E5A 4A 30 0D 33 36 37 34 0D,68
0E62 38 30 14 0D 50 4F 4B 45,20
0E6A 32 30 32 2C 31 36 39 3A,BA
0E72 50 4F 4B 45 32 30 33 2C,AA
0E7A 31 34 3A 50 4F 4B 45 31,A9
0E82 39 36 2C 31 30 34 3A 50,63
0E8A 4F 4B 45 31 39 37 2C 31,40
0E92 35 0D 00 47 30 0D 3F 55,9A
0E9A 53 52 28 26 22 38 42 38,61
0EA2 36 22 2C 30 29 0D 00 84,CF
0EAA FD A9 09 20 A5 89 20 2E,1A
0EB2 83 20 9C 82 A9 10 8D 02,23
0EBA AC 8D 00 AC 20 7B 8C A9,D8
0EC2 00 8D 00 AC 60 0B 76 B3,A5
0ECA 83 BD D3 79 1E F4 A6 F5,DE
0ED2 7B 83 FC B0 10 7C 0C 1F,3F
0EDA 67 CA 7C DE 53 CB C1 7D,26
0EE2 14 64 70 4C 7D B7 EA 51,C9
0EEA 7A 7D 63 30 88 7E 7E 92,69
0EF2 44 99 3A 7E 4C CC 91 C7,6E
0EFA 7F AA AA AA 13 81 00 00,7F
0F02 00 00 A5 B6 48 10 03 20,55
0F0A 36 DD A5 B1 48 C9 81 90,E0
0F12 07 A9 72 A0 D7 20 C5 D8,36
0F1A A9 C7 A4 C5 88 20 C2 DD,56
0F22 68 C9 81 90 07 A9 35 A4,21
0F2A C5 20 06 D6 68 10 03 4C,A9
0F32 36 DD 60 81 49 0F DA A2,71
0F3A 7F 00 00 00 00 05 84 E6,5F
0F42 1A 2D 1B 86 28 07 FB F8,69
0F4A 87 99 68 89 01 87 23 35,5A
0F52 DF E1 86 A5 5D E7 28 83,34
0F5A 49 0F DA A2 A1 54 46 8F,D2
0F62 13 8F 52 43 89 CD C0 72,91
0F6A F0 4A 90 41 C0 76 F0 92,54
0F72 20 80 D9 A9 00 85 16 A5,B6
0F7A C5 48 A9 85 48 A5 C5 48,EB
0F82 A9 B5 48 60 A2 9E A0 00,D1
0F8A 20 8A D9 A9 A7 A0 00 20,64
0F92 58 D9 A9 00 85 B6 A5 C5,E3
0F9A 48 A9 A7 48 A5 16 48 A5,6B
0FA2 C5 48 A9 E7 48 60 A9 9E,F7
0FAA A0 00 4C C5 D8 A9 05 A4,02
0FB2 C5 20 1D D6 20 C2 D9 A9,3E
0FBA 59 A4 C5 A6 BE 20 DD D8,19
0FC2 20 C2 D9 20 82 DA A9 00,F9
0FCA 85 BF 20 09 D6 A9 0A A4,C3
0FD2 C5 20 06 D6 A5 B6 48 10,37
0FDA 0D 20 FF D5 A5 B6 00 09,CC
0FE2 A5 16 49 FF 85 16 20 36,C0
0FEA DD A9 3A A4 C5 20 1D D6,FC
0FF2 68 10 03 20 36 DD A9 3F,92
0FFA A4 C5 4C C2 DD 01,E7
 B0E7
```


MICRO™ is published monthly by:
MICRO INK, Inc.
34 Chelmsford Street
Chelmsford, Massachusetts
617/256-5515

Second Class postage paid at:
Chelmsford, MA 01824

Postmaster: Send address changes to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824

Publication Number:
COTR 395770

Subscription in United States:
$15.00 per year/12 issues

MICRO

Some Useful Memory Locations and Subroutines for OSI BASIC in ROM.

S.R. Murphy
201 N. W. 48th
Seattle, WA 98107

If you want to know more about your OSI BASIC, information is presented which details the use of RAM Scratch Pad Memory and shows where some of the most important Support Subroutines reside.

MICRO has published very little on OSI's BASIC in ROM system. One can only guess that fewer OSI owners are inclined to explore their machines and bend their functions to their own uses in contrast, perhaps, to owners of other 6502 systems. This is a pity because, in contrast to what I read about PET, for example, the BASIC ROM's and the EPROM's that support BASIC, keyboard polling, and the MONITOR are all easily accessed by PEEK or through the MONITOR.

This note may stimulate other OSI BASIC in ROM owners to try some software ideas for custom uses. The following listing of BASIC pointer and subroutine locations make it possible to modify programs written for other MICROSOFT 6502 BASIC interpreters for use with OSI.

MICRO, number 6, pages 49-50, gave a "PARTIAL LIST OF PET SCRATCH PAD MEMORY", by Gary A. Creighton. Since MICROSOFT supplied the BASIC interpreter for both OSI and PET, a principle of parsimony suggests that there should be a strong similarity between the two systems even though OSI uses a more primitive cassette I/O system without the file commands.

Table 1 represents the essence of this similarity in parrallel to the PET table. The notation is essentially the same as Mr. Creighton's except for the use of Hex rather than Decimal.

IND (XY) is an address with the low byte in location $XY and the high byte in location $XY + 1.

M(XY) is the content of memory location $XY.

The description also follows the original with the appropriate modifications for OSI operations. The table is not complete, but, to the best of my knowledge it is accurate.

Finally, in MICRO, number 11, page 37, Don Rindsberg presented an impressive BASIC renumbering program. I have not yet converted the program to OSI because a BASIC renumbering capability is not one of my favorite needs. However, for OSI owners who would like to "roll your own" following Mr. Rindsberg, Table 2 is presented as a substitution for his Table 1 on page 38 that lists the BASIC subroutines needed in his program. The subroutines in Table 2 can, of course, be used for other purposes. $B95E is an excellent Hex to Decimal converter that can be called with a simple machine language program. Similarly, $A77F can be the basis for Decimal to Hex conversion. $A8C3 is a general purpose message printing routine that is easily incorporated into any program. Finally, $A24D makes it relatively simple to modify BASIC programs under computer control.

Table 1
A Partial List of OSI BASIC in ROM Scratch Pad Memory

(Ref. MICRO, No. 6, Pgs. 49 - 50)

IND(01)	Initially, address of cold start ($BD11). Replaced by warm start ($A274).
IND(06)	USR INVAR address.
IND(08)	USR OUTVAR address.
IND(0B)	USR program address.
M(0D)	Number of NULL'S selected.
M(0E)	Terminal character count.
M(0F)	BASIC terminal width.
M(11-12)	Arguments of statements such as PEEK, POKE, GOTO, GOSUB, line numbers, etc.
M(13-5A)	Input buffer.
IND(71)	Scratch pad address for garbage collection, line insertion, etc.
IND(79)	Address of beginning of BASIC code. ($0301)
IND(7B)	Address of beginning of Variable Table.
IND(7D)	Address of first array entry in Variable Table. If no arrays, end of Variable Table.
IND(7F)	Address of end of Variable Table.
IND(81)	Lowest string address.
IND(83)	Scratch pad string address.
IND(85)	Address, plus one, of highest allocated memory.
M(87-88)	Present BASIC line number.
M(89-8A)	Line number at BREAK.
IND(8B)	Pointer to BASIC code for CONT.
M(8D-8E)	Line number for present DATA statement.
IND(8F)	Address of next DATA statement.
IND(91)	Address of next value after comma in present DATA statement.
M(93-94)	ASCII code for present variable.
M(BC-D3)	Subroutine: Points through code one byte at a time, RTS with code value in A and carry clear if ASC(0 - 9); otherwise, carry set. Return A = 0 if end of line. Ignores spaces.
IND(C3)	Code location pointer for above subroutine.
M(AF-B0)	USR input variable storage.
M(FB)	MONITOR keyboard control flag. (= 0 for keyboard).
M(100-107)	Storage of conversion of floating point number to ASCII.
M(1FF)	Top of BASIC stack.
M(200-20E)	Temporary storage for CR simulator subroutine ($BF2D).
M(212)	CTRL C flag. (=$01 if CRTL C off).
M(213-216)	Temporary storage, keyboard polling program (#FD00).

Table 2
OSI BASIC Routines Needed for BASIC Renumbering

(Ref. MICRO, No. 11, Pg. 38)

$A24C	Print an error message from the message table. Enter with X containing the location of the message relative to $A164. Message terminator is ASCII having bit 7 on.
$A24D	BASIC line insertion routine. Enter with line assembled in the line buffer $0013-$005A with 00 as line terminator. Also, character count must be in $005D and the line number(hex) at $0011/12.
$A77F	Evaluate an expression whose beginning address is in $00C3/C4. Use this subroutine to convert from ASCII to binary, with the result appearing in the floating accumulator: $00AC/AD/AE/AF.
$B7E8	Convert fixed number in $00AD/AE to floating number. Enter with the result appearing in the floating accumulator: $00AC/AD/AE/AF.
$B408	Convery binary value, such as line number, in floating accumulator to two-byte fixed number and place in $0011/12.
$B96E	Convert floating number at $00AC/AD/AE/AF to ASCII and place in string starting at $0101, preceded by a space or minus sign at $0100 and terminated by 00.
$A274	BASIC warm start. Prints "OK".
$A8C3	Prints message. Enter with ADH in Y, ADL in A. Message is ASCII string ending with 00.
$B95E	Print the decimal integer whose hex value is in registers A and X, for example, a line number.

A Tape Indexing System for the PET

A solution is provided for the PET cassette tape problem. Using inherent capabilities of the PET, a procedure is presented which permits use of the recorder's fast forward and fast rewind facilities to rapidly index and portion of the tape.

Alan R. Hawthorne
611 Vista Drive
Clinton, TN 37716

A frustrating problem for PET owners occurs when it becomes necessary to load a program or read a data file that has been written in the middle of a cassette tape. Since Commodore chose not to include a cassette recorder with an index counter, the user is left with the following options:

1) Load only one program per tape. (This can make personal computing unnecessarily expensive for the hobbist with hundreds of programs.)

2) Let the PET slowly search through the tape until it finds the correct file name. (This process is much too slow, since it can take up to 30 minutes to search one side of a 60 minute tape.)

3) Guess where the program might be on the tape and run the tape to this point using the fast forward speed on the recorder; then let the PET begin to search for the program. (This guessing is no fun. One often runs past the desired program and wastes even more time).

I decided that there must be a better way to use the PET for reading multiple files. An index finder on the recorder would, in essence, permit me to use option 3 above, but with the guesswork reomved in positioning with the fast forward speed. I contemplated implementing a photodetector and an LED as an index counter, but this would require modification of the recorder plus hardware for counting and displaying. A much simpler solution would be to develope a software index counter that would take advantage of existing recorder switch-sensing and motor-control capabilities of the PET. The machine language program, described below, uses an index number corresponding to each program position on the tape. Also given is the methoc for determining the correct index number corresponding to each program on the tape.

Indexing Approach: Theory

A successful tape positicning program can be implemented if the fast forward speed of the recorder is run for the correct length of time and if this correct length of time can be determ ned for a given program.

The first requirement can be met easily by using the PET's ability to sense if the recorder buttons are pressed and to stop the recorder under program control. The tape of interest is simply loaded into the recorder and rewound; the correct time constant for the desired program is then entered into the PET. The positioning program instructs the user to press "fast forward." The program waits until a recorder button depression is detected, then begins timing until a time corresponding to the index time has elapsed, at which point the recorder motor is stopped. A prompt character is output to the PET screen indicating that the tape is positioned at the beginning of the desired program on tape and that the user should now pres the "stop" button. Upon sensing the depression of the recorder stop button, the indexing program places the recorder under manual control for subsequent use in loading the program from tape and then exits to the operating systems monitor. If this machine language positioning program is stored in a safe memory such as the cassette number 2 buffer (M826-M1023) when the PET is powered up, it will always be available for positioning programs and data files with no time lost in loading the program each time it is needed.

The second requirement for implementing an indexing system, that of determining the correct time constant for a given program, is more demanding. Not being able to read the tape header while the recorder is running fast forward, one must find another means of determining the fast forward time required for positioning a tape. A related time that can be obtained easily is the amount of time required to rewind the program tape. The PET can detect when the rewind button is depressed and can count time until the user presses "stop" when the tape is rewound. This time can be directly, although not simply, correlated with the fast forward time required to position to the beginning of the program to be entered. Of course the problem in relating the rewind speed to the fast forward speed occurs because the tape speed (cm/sec pass the recorder head) varies even though the drive speed (revolutions/sec) is the same in each direction. (That the drive speed is the same fast forward as rewind is easily proven by measuring the time taken to run through a complete tape in the fast forward mode. This time can be compared with the time taken to rewind the tape. The two times should be approximately equal.)

With the forward and reverse drive speeds the same, the following integral equation can be used to relate the rewind time (tr) and the fast forward time (tf) in terms of the minimum tape radius (ro), the rate of radius change (c), and the time (tm) required to rewind the tape from the end to the beginning.

$$\int_0^{t_f} (r_0 + ct)dt = \int_0^{t_r} [r_0 + c(t_m - t)]dt$$

This equation can be solved for the fast forward speed:

Listing 1.

```
970   20 D0 D6           JSR     54992
973   A9 10              LDAIM   16
975   2C 10 E8  WAIT1    BIT     59408
978   D0 FB              BNE     WAIT1
980   A0 02 02  NEXT     LDA     514
983   CD 02 02  WAIT2    CMP     514
986   F0 FB              BEQ     WAIT2
988   C6 08              DEC     08
990   D0 F4              BNE     NEXT
992   C6 09              DEC     09
994   10 F0              BPL     NEXT
996   A9 34              LDAIM   52
998   8D 07 02           STA     519
1001  A9 3D              LDAIM   61
1003  8D 13 E8           STA     59411
1006  A9 5F              LDAIM   95
1008  20 D2 FF           JSR     65490
1011  A9 10              LDAIM   16
1013  2C 10 E8  WAIT3    BIT     59408
1016  F0 FB              BEQ     WAIT3
1018  A9 00              LDAIM   00
1020  8D 07 02           STA     519
1023  60                 RTS
```

$$t_f = (2t_m t_r + k^2 + 2kt_r - t_r^2)^{1/2} - k$$

where

$$k = \frac{t_m t_r - \frac{1}{2}t_r^2 - \frac{1}{2}t_f^2}{t_f - t_r}$$

The value for k can be determined for tapes of various lengths (15 min., 30 min., 60 min., etc.) by running fast forward for a time, measuring the rewind time, and evaluating equation 3. This should be repeated for several different times and an average value obtained for k.

Program Implementation

Using the techniques outlined, a tape indexing program can easily be implemented for the PET. Listing 1 gives a machine language program that will run a tape fast forward for a given time and stop the cassette motor. The program is run, after the correct tape has been loaded and rewound, by calling the user function X = USR (TC) where TC, the index time constant, is the number of jiffies required to position the tape correctly. Time is evaluated in jiffies because the PET has a jiffy counter which is convenient to use and the timing resolution provided is quite sufficient.

The program uses several features of the PET's operating system. The subroutine at M54992 converts the argument of the USR function from the floating accumulator to a 16-bit integer with the LSB in M8 and the MSB in M9. Bit 4 of M59408 senses the status of the recorder switches. If any switch of the recorder is pressed, the content of this bit is 0; otherwise it is 1.

The jiffy counter, which the PET uses as a part of its real-time clock, is located in M514 and is incremented 60 times a second by the operating system. The cassette flag is located in M519. A 52 must be loaded in order to control the recorder motor using the program and then restored to 0 before exiting the program, leaving the recorder under manual control. With the cassette flag set correctly, the recorder can be stopped by the program by loading the value 61 into M59411. Finally, the subroutine at M65490 is used to display a prompt on the video screen informing the user that the tape is positioned and that the "stop" button should be pressed

The positioning program can be called either from a BASIC program or by direct command. Listing 2 is a BASIC program for loading the machine language program into memory when the PET is powered up. The program is stored in the upper portion of the number 2 cassette buffer and will remain loaded until the user writes over the memory or the PET is reset. This location leaves available protected memory from M826 to M970 for other machine language programs.

Listing 3 is a BASIC program called TAPE capable of providing several useful functions for a tape indexing system. The program, as currently dimensioned, indexes 10 tapes with up to 10 programs per tape. The functions are available by entering various commands. To position a tape for reading program number k on tape number ℓ, enter R. (The machine language program of Listing 1 is assumed to be loaded.) To update a program name or index time constant in the index, enter a U. The tape number and program number will be requested by the program.

To determine the rewind time and fast forward time for a program number k, enter a T. The tape containing the program to be indexed should be positioned so that it is at the end of the program. If the tape is not at this point it can be positioned by verifying, using the program name (i.e., VERIFY "program name"). This will position the tape correctly, even though a verify error will occur. The time constant measured and displayed using the T command is actually the index time for the program k + 1 and is automatically entered into the index by the T command, so that the U command is not needed.

To look at the index of a given tape, enter I and the tape number. The index will appear on the PET screen with the program number, name, and time constant displayed. To save the index data file, an S command is entered. The index file should be saved if any tape index was updated or added to by using the T command. The data file is placed directly following the BASIC program TAPE on the tape. This is done by verifying TAPE before writing the data file. If the index data file has never been written on tape, the TAPE program should be entered at 10200 (i.e., RUN 10200) instead of 10000 since the first thing the program does is read the data file.

The most important part of the TAPE program is the index time constant

Listing 2.

```
00  REM  TAPE POSITIONING PROGRAM,  X=USR(TC)
10  DATA  32, 208, 214, 169, 16, 44, 16, 232, 208
20  DATA  251, 173, 2, 2, 205, 2, 2, 240, 251
30  DATA  198, 8, 208, 244, 198, 9, 16, 240, 169
40  DATA  52, 141, 7, 2, 169, 61, 141, 19, 232
50  DATA  169, 95, 32, 210, 255, 169, 16, 44, 16
60  DATA  232, 240, 251, 169, 0, 141, 7, 2, 96
70  FOR  A=970 TO 1023: READ B : POKE A,B : NEXT
80  POKE  1, 202, : POKE 2,3
90  END
```

determining routine. In order to use the machine language positioning program, all that is needed is the time constant.

If one simply writes the time constant by the program name on his tape label, there is no need for the TAPE program to be used each time a specific program is to be read. Instead, TAPE will most likely be read when index editing or surveying is desired. The pertinent lines for obtaining the index time constant are 14100-14700. The values determined for tm and k in equation 2 were 6000 jiffies and 5000 jiffies respectively, for a 60 minute tape.

Although the constants for a 30 minute tape were somewhat larger than half the 60 minute tape constants, the relatively low degree of accuracy required to position within the 10 second buffer written by the PET prior to each program allows considerable freedom in the selection of the constants. Line 14400 uses the PET BASIC function WAIT to monitor the recorder buttons in measuring the rewind time. The user should try to press "stop" as soon as the tape is rewound, since considerable error can be introduced if the rewind time is not measured consistently.

Final Comments

Perhaps a word of caution is in order. The user should avoid placing programs that may require extensive revisions in the middle portion of a tape, since the revised program might then extend on to the next program on the tape. However, once a program has been developed, the use of multiple files per tape is often quite convenient.

After implementing the tape indexing and positioning programs, I find that I no longer dread the thought of having to read a program from the middle of a cassette. In fact, reading the seventh or eighth program on the tape takes only slightly longer than reading the first program. Hopefully, other PET enthusiasts will find the program useful. In any case, discovering and utilizing some of the "hidden" powers of my PET was half the fun.

Listing 3.

```
10000 DIM TN$(10,10),TM(10,10)
10050 OPEN1,1,0,"TAPE INDEX"
10070 FORJ=1TO10
10100 FORI=1TO10:INPUT#1,TN$(J,I),T$:TM(J,I)=VAL(T$):NEXT:NEXT
10150 CLOSE1
10200 PRINT"R:READ,U:UPDATE,T:TIME,I:INDEX,S:SAVE"
10250 PRINT"TAPE # & COMMAND":INPUTL,C$
10300 IFC$="R"THENGOSUB12000:GOTO10200
10400 IFC$="U"THENGOSUB13000:GOTO10200
10500 IFC$="T"THENGOSUB14000:GOTO10200
10600 IFC$="I"THENGOSUB15000:GOTO10200
10700 IFC$="S"THENGOSUB11000:GOTO10200
10800 PRINT"???":GOTO10200
11000 PRINT"TAPE REWOUND":INPUTY$
11100 VERIFY"TAPE":WAIT59408,16
11200 POKE243,122:POKE244,2:OPEN1,1,"TAPE INDEX"
11300 FORJ=1TO10
11400 FORI=1TO10:T$=STR$(TM(J,I)):PRINT#1,TN$(J,I)","T$:NEXT
11500 NEXT
11600 CLOSE1:RETURN
12000 PRINT"ENTER PGM # ":INPUTK
12100 PRINT"TAPE ";L;" LOADED & REWOUND":INPUTY$
12200 PRINT"PRESS F-F":X=USR(TM(L,K))
12300 RETURN
13000 PRINT"ENTER PGM # TO UPDATE (0 TO EXIT)":INPUTK
13100 IFK=0THENRETURN
13200 PRINT"NEW TITLE":INPUTTN$(L,K)
13300 PRINT"NEW TIME":INPUTTM(L,K)
13400 GOTO13000
14000 PRINT"PGM # & TITLE":INPUTK,TN$(L,K)
14100 PRINT"ENTER 1 FOR 30 MIN TAPE, 2 FOR 60 MIN"
14200 INPUTZ:MX=3000*Z:TK=2500*Z
14300 PRINT"PRESS REWIND"
14400 WAIT59408,16,16:T=TI:WAIT59408,16
14500 T=TI-T:PRINT"REWIND TIME = ";T
14600 TM(L,K+1)=INT(SQR(2*MX*T+TK↑2+2*TK*T-T↑2)-TK)
14700 PRINT"FAST FORWARD TIME = ";TM(L,K+1)
14800 RETURN
15000 PRINT"♥","***TAPE ";L;" INDEX***":PRINT
15100 FORI=1TO10:PRINT"#";I;TN$(L,I);TAB(32);TM(L,I):PRINT:NEXT
15200 RETURN
```


More Classifieds on page 20.

Subroutine Parameter Passing

Mark Swanson
177 Hastings Mill Road
Streamwood, IL 60103

A technique that makes it easy to pass parameters to subroutines is presented. While this method has been known and used for many years on the big computers, it may be new and useful to many microcomputerists.

Passing information from a main program to a subroutine is usually done by either pushing it on the stack or storing the information in an area common to both routines. An alternative method involved having the parameters after the subroutine call.

When a jump subroutine is executed the return address is stored on the stack and control is passed to the subroutine. If we put our parameters after the jump subroutine instruction, the return address on the stack will now point to this data. The subroutine can now pull the return address off the stack, fetch the parameters useing the return address, increment the return address to skip over the parmeters, and use this new address to return to the calling program.

Here is an example of using this method of parameter passing to print a character string. The program MAIN contains a jump subroutine to the subroutine PTRSTR. The address of the beginning of the string follows the JSR instruction. The end of the string is marked by a zero byte. The routine first references the stack to get the return address and stores this in a temporary zero page location (locations zero and one). Using this address we now can access the string starting address located after the JSR. The string starting addrss is moved into a temporary zero page location (locations two and three). Using indirect indexed addressing we load a byte of the character string, call a routine which prints a single byte, increments the Y register, and loops until the zero byte is found. After the entire string is printed, we increment the return address by two to skip over the string address parameters. We can now return to the calling program via an indirect jump to the temporary return address locations (locations zero and one).

This method of parameter passing can be very useful when dealing with subroutines that are called frequently or which pass large amounts of data.

```
MAIN PROGRAM        .
                    .
                    JSR    PRTSTR    JUMP TO SUBROUTINE TO PRINT A STRING
                    =      $78       LOW PORTION OF STRING ADDRESS
                    =      $56       HIGH PORTION OF STRING ADDRESS
                    .                RETURN HERE TO CONTINUE PROCESSING
                    .
5678                =      "PRINT THIS MESSAGE"
                    =      $00       HEX ZERO TO MARK END OF STRING
```

```
ASSEMBLE LIST

                      0100 ;MOVE TBL 1 TO TBL2
                      0110            .BA  $400
0400-- A/  0B         0120 LOOP       LDY  #00
0402-- B9  0B  04     0130            LDA  TBL1,Y
0405-- 89  0B  05     0140            STA  TBL2,Y
0408-- C8             0150            INY
0409   D0  F7         0160            BNE  LOOP
                      0170 ;
040B                  0180 TBL1       .DS  256
050B                  0190 TBL2       .DS  256
                      0200 ;
                      0210                 .EN

LABEL FILE    1 = EXTERNAL

START = 0400          LOOP = 0402          TBL1 = 040B
TBL2 = 050B
110000,060B,060B
```

```
                         PRINT STRING SUBROUTINE
                         MARK SWANSON

                         SUBROUTINE TO PRINT A STRING OF CHARACTERS

023A            ZERO   *     $00E0
023A            ONE    *     $00E1
023A            TWO    *     $00E2
023A            THREE  *     $00E3
023A            STACK  *     $0100
023A            PUTCHR *     $1234  SOME PRINT CHARACTER SUBROUTINE

0200                   ORG   $0200

0200 D8         PRTSTR CLD          CLEAR DECIMAL MODE
0201 BA                TSX          TRANSFER STACK PTR TO X REG

                SINCE POINTER ALWAYS POINTS TO NEXT POSITION
                AVAILABLE, INCREMENT BY ONE

0202 E8                INX
0203 BD 00 01          LDAX  STACK  LOAD LOW PORTION OF RETURN ADDRESS
0206 85 E0             STA   ZERO   SAVE
0208 E8                INX          INCR X TO NEXT STACK ENTRY
0209 BD 00 01          LDAX  STACK  LOAD HIGH PORTION OF RETURN ADDRESS
020C 85 E1             STA   ONE    SAVE
020E 9A                TXS          RESET STACK POINTER
020F E6 E0             INCZ  ZERO   ADDRESS OFF BY 1, SO NOW WE
0211 D0 02             BNE   OVER   INCREMENT IT
0213 E6 E1             INC   ONE    HIGH ADDRESS TOO, IF NECESSARY

0215 A0 00      OVER   LDYIM $00    ZERO Y REGISTER
0217 B1 E0             LDAIY ZERO   LOAD FIRST PART OF STRING ADDRESS
0219 85 E2             STA   TWO    SAVE
021B C8                INY          BUMP POINTER
021C B1 E0             LDAIY ZERO   LOAD SECOND PART OF STRING ADDRESS
021E 85 E3             STA   THREE  SAVE

                SUBROUTINE NOW HAS THE STRING ADDRESS
                NOW PRINT STRING UNTIL A HEX 00 IS FOUND

0220 A0 00             LDYIM $00    ZERO Y REGISTER
0222 B1 E2      LOOP   LDAIY TWO    LOAD A CHARACTER OF STRING
0224 F0 06             BEQ   FINISH IF EQUAL TO ZERO, FINISHED
0226 20 34 12          JSR   PUTCHR SOME SUBROUTINE TO PRINT A CHARACTER
0229 C8                INY          INCREMENT POINTER
022A D0 F6             BNE   LOOP   UNCONDITIONAL

022C 18         FINISH CLC          CLEAR CARRY
022D A5 E0             LDA   ZERO   INCREMENT RETURN ADDRSSS BY
022F 69 02             ADCIM $02    TWO TO SKIP OVER
0231 85 E0             STA   ZERO   STRING ADDRESS PARAMETERS
0233 90 02             BCC   END    DONE IF NO CARRY
0235 E6 E1             INC   ONE    BUMP HIGH ADDRESS IF CARRY
0237 6C E0 C0   END    JMI   ZERO   JUMP INDIRECT TO RESUME MAIN PROGRAM
```

APPLE II Hires Picture Compression

Every APPLE owner is aware of the wonderful pictures that can be made with the HIRES graphics. A very interesting technique is presented which allows greater efficiency in encoding picture information, and which leads to some additional special effects.

Bob Bishop
213 Jason Way
Mountain View, CA 94043

Almost every APPLE II owner has, by now, seen examples of how the APPLE II can display digitized photographs in its HIRES graphics mode. These images consist of 192 x 280 arrays of dots all of the same intensity. By clustering these dots into groups (such as in "dithering"), it is even possible to produce pictures having the appearance of shades of gray. Several "slide shows" of these kinds of pictures have been created by both Bill Atkinson and myself and are available through various sources, such as the Apple Software Bank. A typical "slide show" consists of about 11 pictures on a standard 13-sector disk. Why only 11 pictures? Because that's about all that will fit on a 13-sector disk.

Each HIRES picture must reside in one of the two HIRES display areas before it can be seen. The first area, \$2000-\$3FFF, is called the *primary* display buffer; the second area, \$4000-\$5FFF, is called the *secondary* display buffer. It is obvious that each of these display areas are 8-K bytes long. Consequently, HIRES pictures are usually stored as 8-K blocks of data, exactly as they appear in a display buffer. But do they have to be stored that way?

If you look closely at a HIRES picture, you can almost always detect small regions that look very similar to other small regions elsewhere in the picture. For example, HIRES displays usually contain regions of pure white or pure black. In the case of dithered pictures, the illusion of gray may be caused by micro-patterns of dots that are similar to other gray patterns somewhere else. Clearly, HIRES pictures tend to contain a lot of redundancy. If there was some way of removing this redundancy then it would be possible to store HIRES pictures in less than the customary 8-K bytes of memory.

Suppose we were to divide the display into small rectangular clusters, each 7 bits wide, by 7 bits high. Then a picture would consist of 24 rows of these picture elements ("pixels"), with 40 of them per row. (Note the resemblance to the APPLE II's TEXT mode of 24 lines, 40 columns per line!) The total number of pixels that would be needed to define a HIRES picture would then be 40 times 24, or 960. However, not all 960 pixels would be unique if there was redundancy in this picture.

To try out these ideas, I used Atkinson's LADY BE GOOD picture (from the Apple Magic Lantern — Slide Show 2) shown in Figure 1, and wrote a program to extract all the different pixels. I found that only 662 of the 960 pixels were unique. This meant that almost one third of the picture was redundant!

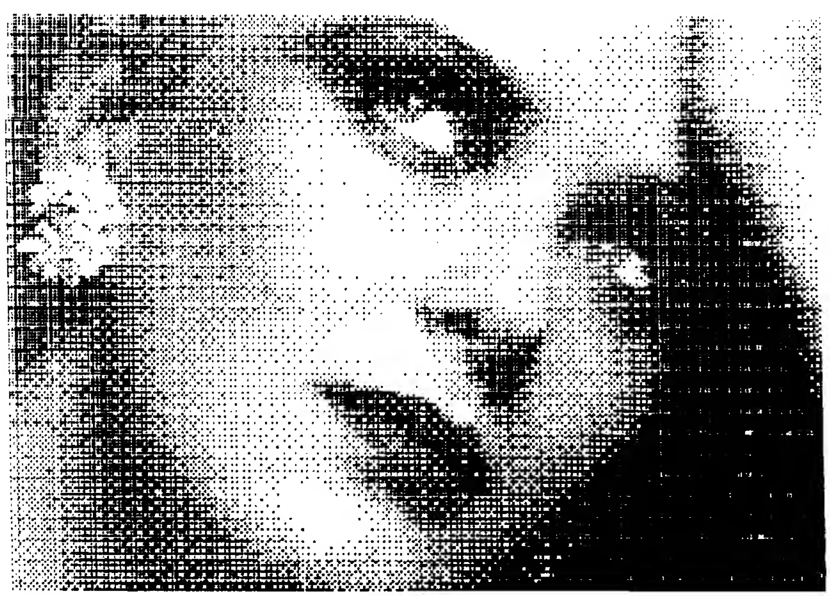


Figure 1: **(Max errors/pixel = 0)**

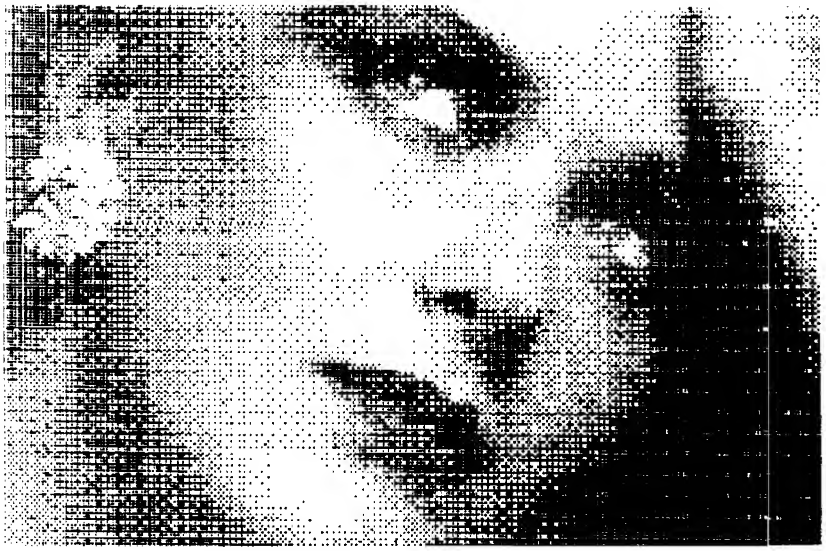


***Figure 2:* (Max errors/pixel = 3)**

The next question that came to mind was: of the 622 unique pixels, how 'unique' were they? Was it possible that there might be two or more pixels that were almost the same, except for maybe one or two dots that differed? If so, then it could be possible to regard these as being identical 'for all practical purposes' since the error in the resulting picture would hardly be noticed.

To examine this possibility, I modified my program to extract only those pixels that differed by more than a specified MAX ERRORS/PIXEL. Table 1 shows the results. If we allow, at most, 1 dot to be wrong in any one pixel, then we need only 492 pixels to define the picture, which is only about half of the original 960 pixels! As we allow more and more errors per pixel, the number of pixels required to reconstruct the picture decreases accordingly, until we reach 28 errors/pixel. At this point we are allowing half of the dots to be wrong. Since total black and total white are always included in every pixel set (to prevent black or white areas from becoming dotted), pictures with MAX ERRORS/PIXEL greater than or equal to 28 can always be composed of no more than two pixels, namely the black and white pixels.

Suppose we now try to reconstruct the original picture from our extracted pixel set. Clearly, the fewer pixels we have available for synthesizing, the poorer the result will be. Figures 2 through 5 show the results of synthesizing LADY BE GOOD with MAX ERRORS/PIXEL of 3, 7, 14, and 28. The number of pixels used in each case was 245, 75, 15, and 2, respectively. Notice that the difference in quality between Figures 1 and 2 is not all that objectionable. The advantage that Figure 2 has is that it can be stored in less than 3-K bytes of memory! (245 pixels at 8 bytes/pixel, plus 960 bytes to define which pixels go where.)

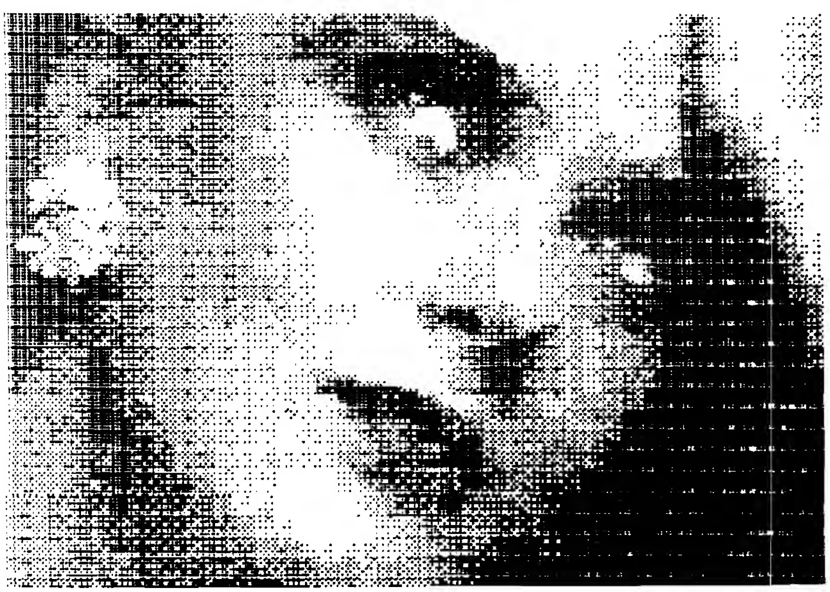


***Figure 3:* (Max errors/pixel = 7)**

Thus it is clearly possible to store an 8-K HIRES picture in considerably less than 8-K bytes, if you are willing to accept a little loss in the image quality. By using this principle, I have produced a "Super Slide Show" containing 33 pictures on a single disk. (Copies may be obtained from Apple's Software Bank.)

```
SYMBOL TABLE
BILD    0B00
BLUP    0B11
LUPE    0B17
NEXT    0B2C
OVER    0B31
GOOD    0B3F
RCON    0B80
RLUP    0B96
LOOP    0BA4
CONT    0BB7
INC     0BC3
SEND    0BD3
MUTO    0BF1
RET     0BFF
MOVE    1100
MLUP    1107
COMP    1123
CLUP    112E
PREP    1154
INIT    1193
STOR    1200
X40     122C
XAT     0000
YAT     0001
ZAT     0002
XTO     0003
YTO     0004
ZTO     0005
SCOR    0006
XMAX    0007
YMAX    0008
XTMP    0009
YTMP    000A
BEST    000B
AT      000C
TO      000E
ERR     0010
XIN     0011
YIN     0012
PROD    0013
HGRL    0C00
HGRH    0D00
BITS    1000
BELL    FF3A
END     1261
```

The Compression Program

Listings 1 and 2 show the compression routines (and some associated data tables), and require an APPLE II with at least 32-K bytes of memory. The routines consist of two basic parts—the "analysis" portion, and the "synthesis" portion.

The analysis routine ($0B00) searches the primary HIRES display buffer ($2000-$3FFF) and compares each pixel there with the pixels in its own current pixel table (which starts at $0600) looking for a "match". If it finds a pixel in the table that matches to within the specified MAX ERRORS/PIXEL (location $10), it calls a match and proceeds to the next pixel in the picture. If it fails to find a match, it adds the pixel to its current pixel table and then proceeds.

The synthesis routine ($0B80) works in the other direction. It first compares each pixel of the primary buffer with each pixel in the pixel table to find the best match. It then places this pixel in the corresponding location in the secondary HIRES buffer, thus synthesizing the best approximation to the primary picture as it can by using the pixels in its pixel table.(Since the analysis routine doesn't know where its pixel table originated, it is possible to synthesize one picture from another picture's pixels! The result is usually surprisingly good.)

The routines are very easy to use. Simply load the picture to be compressed into $2000-$3FFF, set MAX ERRORS/PIXEL into $10, and then call the routine at $0B00. When the routine returns, locations $07 and $08 contain the number of extracted pixels in the form: NUMBER = 1 + (contents of $07) + 40* (contents of $08).

To synthesize the picture from the extracted pixels, simply call the routine at $0B80. When the routine returns, the reconstructed picture will be in the secondary HIRES buffer ($4000-$5FFF).

If you have a 48-K APPLE and a disk, you can use the BASIC program shown in Listing 3. This program calls the compression routines (Listings 1 and 2) in a more user-oriented way so that they are even easier to use. The program displays a menu of options that let you:

L—Load a picture from disk into the primary HIRES buffer
1—Display the picture currently in the primary HIRES buffer
2—Display the picture currently in the secondary HIRES buffer
A—Analyze the primary picture (create the pixel table.)
S—Synthesize the primary picture using the current pixel table.
D—Issue disk commands.
X—Transfer the compressed picture to disk drive number 2.

Figure 4: **(Max errors/pixel = 14)**

None of the selections require you to hit RETURN; just hit the corresponding character. When specifying "L", the program will ask you for the name of the file to be loaded. When specifying "A", you will be asked for the minimum error per pixel that you will allow. (This does require a RETURN.). The "D" command will give a colon (:) as the prompt character and will allow you to issue disk commands. It will continue in this mode until you give it a null command (hit RETURN) at which time it will return to the menu. The "X" command saves the compressed picture (960 bytes) and its corresponding pixel table (up to 2K bytes) onto a disk file. (I will leave it up to the interested reader to figure out how to "un-compress" this data.)

Concluding Remarks

While the methods in this paper work pretty well, they may not represent the optimum way of compressing APPLE II picture data. For example, my choice of 7 x 8 dot pixels was somewhat arbitrary. Is it possible to get better compression ratios by choosing smaller (or larger) pixel sizes?

Another interesting question is: Given a picture that was reconstructed from a given set of N pixels, is it possible to find another set of N pixels that gives a better result?

I hope that these unanswered questions will help motivate someone else into joining the investigation of HIRES picture compressing methods.

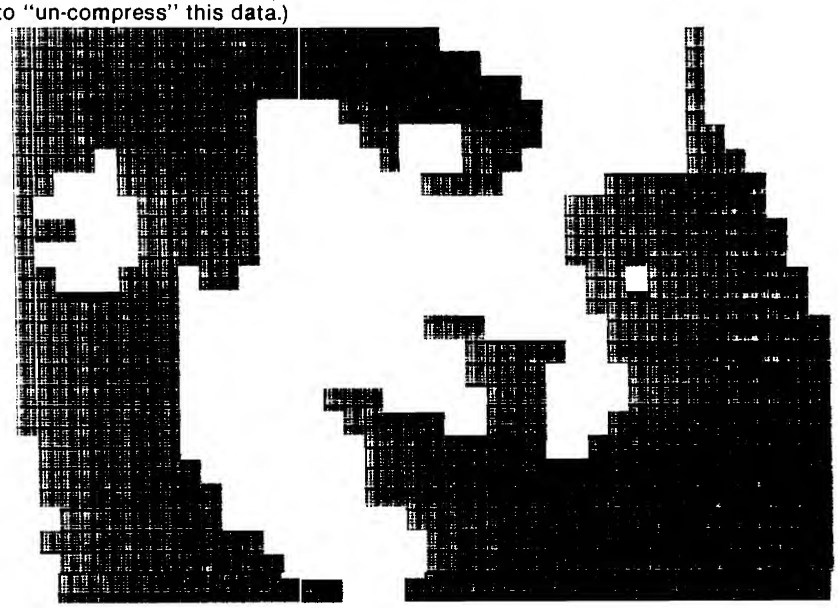
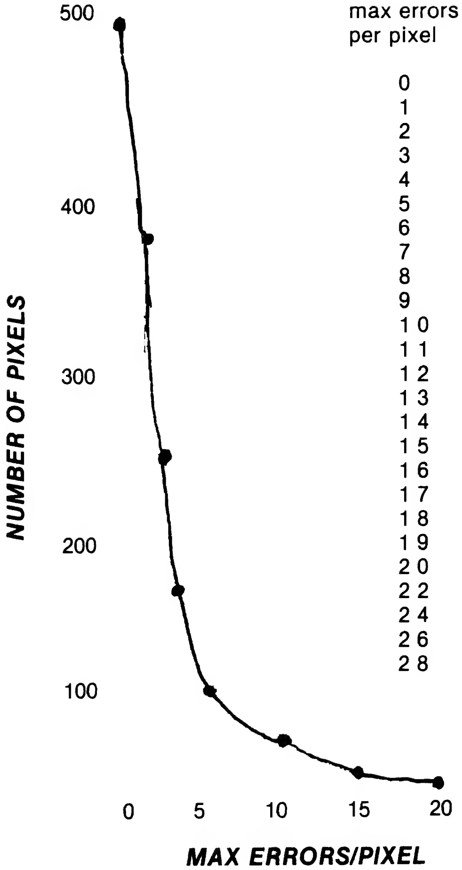


Figure 5: **(Max errors/pixel = 28)**



max errors per pixel	number of pixels
0	662
1	492
2	334
3	245
4	178
5	135
6	100
7	75
8	53
9	44
1 0	34
1 1	29
1 2	24
1 3	18
1 4	15
1 5	11
1 6	10
1 7	7
1 8	7
1 9	6
2 0	5
2 2	4
2 4	4
2 6	4
2 8	2

:A

Listing 1.

```
                 :BUILD PIXEL TABLE
                 :
                        .OR 0B00
0B00  209311     BILD   JSR INIT
0B03  A900              LDA 00
0B05  8500              STA *XAT
0B07  8501              STA *YAT
0B09  A901              LDA 01
0B0B  8502              STA *ZAT
0B0D  A903              LDA 03
0B0F  8505              STA *ZTO
0B11  A900       BLUP   LDA 00
0B13  8503              STA *XTO
0B15  8504              STA *YTO
0B17  202311     LUPE   JSR COMP
0B1A  A510              LDA *ERR
0B1C  C506              CMP *SCOR
0B1E  B01F              BCS GOOD
0B20  A503              LDA *XTO
0B22  C507              CMP *XMAX
0B24  D006              BNE NEXT
0B26  A504              LDA *YTO
0B28  C508              CMP *YMAX
0B2A  F005              BEQ OVER
0B2C  20F10B     NEXT   JSR NUTO
0B2F  D0E6              BNE LUPE
0B31  20F10B     OVER   JSR NUTO
0B34  200011            JSR MOVE
0B37  A503              LDA *XTO
0B39  8507              STA *XMAX
0B3B  A504              LDA *YTO
0B3D  8508              STA *YMAX
0B3F  E600       GOOD   INC *XAT
0B41  A500              LDA *XAT
0B43  C928              CMP 28
0B45  D0CA              BNE BLUP
0B47  A900              LDA 00
0B49  8500              STA *XAT
0B4B  E601              INC *YAT
0B4D  A501              LDA *YAT
0B4F  C918              CMP 18
0B51  D0BE              BNE BLUP
0B53  4C3AFF            JMP BELL
                 :
                 :RECONSTRUCTION
                 :
                        .OR 0B80
0B80  A900       RCON   LDA 00
0B82  8D50C0            STA $C050
0B85  8D52C0            STA $C052
0B88  8D55C0            STA $C055
0B8B  8D57C0            STA $C057
0B8E  8503              STA *XTO
0B90  8504              STA *YTO
0B92  A903              LDA 03
```

```
0B94  8502    0560       STA *ZAT
0B96  A9FF    0570  RLUP LDA 0FF
0B98  850B    0580       STA *BEST
0B9A  A900    0590       LDA 00
0B9C  8500    0600       STA *XAT
0B9E  8501    0610       STA *YAT
0BA0  A901    0620       LDA 01
0BA2  8505    0630       STA *ZTO
0BA4  202311  0640  LOOP JSR COMP
0BA7  A506    0650       LDA *SCOR
0BA9  C50B    0660       CMP *BEST
0BAB  B00A    0670       BCS CONT
0BAD  850B    0680       STA *BEST
0BAF  A500    0690       LDA *XAT
0BB1  8509    0700       STA *XTMP
0BB3  A501    0710       LDA *YAT
0BB5  850A    0720       STA *YTMP
0BB7  A500    0730  CONT LDA *XAT
0BB9  C507    0740       CMP *XMAX
0BBB  D006    0750       BNE INC
0BBD  A501    0760       LDA *YAT
0BBF  C508    0770       CMP *YMAX
0BC1  F010    0780       BEQ SEND
0BC3  E600    0790  INC  INC *XAT
0BC5  A500    0800       LDA *XAT
0BC7  C928    0810       CMP 28
0BC9  D0D9    0820       BNE LOOP
0BCB  A900    0830       LDA 00
0BCD  8500    0840       STA *XAT
0BCF  E601    0850       INC *YAT
0BD1  D0D1    0860       BNE LOOP
0BD3  A509    0870  SEND LDA *XTMP
0BD5  8500    0880       STA *XAT
0BD7  A50A    0890       LDA *YTMP
0BD9  8501    0900       STA *YAT
0BDB  A902    0910       LDA 02
0BDD  8505    0920       STA *ZTO
0BDF  200012  0930       JSR STOR
0BE2  200011  0940       JSR MOVE
0BE5  20F10B  0950       JSR NUTO
0BE8  A504    0960       LDA *YTO
0BEA  C918    0970       CMP 18
0BEC  D0A8    0980       BNE RLUP
0BEE  4C3AFF  0990       JMP BELL
              1000
0BF1  E603    1010  NUTO INC *XTO
0BF3  A503    1020       LDA *XTO
0BF5  C928    1030       CMP 28
0BF7  D006    1040       BNE RET
0BF9  A900    1050       LDA 00
0BFB  8503    1060       STA *XTO
0BFD  E604    1070       INC *YTO
0BFF  60      1080  RET  RTS
              1090  :
              1100  :MOVE A PIXEL
              1110  :FROM XAT,YAT,ZAT
              1120  :TO XTO,YTO,ZTO
              1130  :
              1140       .OR 1100
1100  8A      1150  MOVE TXA
1101  48      1160       PHA
1102  98      1170       TYA
1103  48      1180       PHA
1104  205411  1190       JSR PREP
1107  A400    1200  MLUP LDY *XAT
1109  B10C    1210       LDA (AT   ),Y
110B  A403    1220       LDY *XTO
110D  910E    1230       STA (TO   ),Y
110F  A50D    1240       LDA *AT   +01
1111  6904    1250       ADC 04
1113  850D    1260       STA *AT   +01
1115  A50F    1270       LDA *TO   +01
1117  6904    1280       ADC 04
1119  850F    1290       STA *TO   +01
111B  CA      1300       DEX
111C  D0E9    1310       BNE MLUP
111E  68      1320       PLA
111F  A8      1330       TAY
1120  68      1340       PLA
1121  AA      1350       TAX
1122  60      1360       RTS
              1370  :
              1380  :COMPARE PIXEL
              1390  :AT XAT,YAT,ZAT
              1400  :TO XTO,YTO,ZTO
              1410  :
1123  8A      1420  COMP TXA
1124  48      1430       PHA
1125  98      1440       TYA
1126  48      1450       PHA
1127  205411  1460       JSR PREP
112A  A900    1470       LDA 00
112C  8506    1480       STA *SCOR
112E  A400    1490  CLUP LDY *XAT
1130  B10C    1500       LDA (AT   ),Y
1132  A403    1510       LDY *XTO
1134  510E    1520       EOR (TO   ),Y
1136  297F    1530       AND 7F
1138  A8      1540       TAY
1139  B90010  1550       LDA BITS,Y
113C  6506    1560       ADC *SCOR
113E  8506    1570       STA *SCOR
1140  A50D    1580       LDA *AT   +01
1142  6904    1590       ADC 04
1144  850D    1600       STA *AT   +01
1146  A50F    1610       LDA *TO   +01
1148  6904    1620       ADC 04
114A  850F    1630       STA *TO   +01
114C  CA      1640       DEX
114D  D0DF    1650       BNE CLUP
114F  68      1660       PLA
1150  A8      1670       TAY
```

```
1151  68      1680       PLA
1152  AA      1690       TAX
1153  60      1700       RTS
              1710  :
1154  A502    1720  PREP LDA *ZAT
1156  6A      1730       ROR
1157  6A      1740       ROR
1158  6A      1750       ROR
1159  6A      1760       ROR
115A  2960    1770       AND 60
115C  850D    1780       STA *AT  +01
115E  A505    1790       LDA *ZTO
1160  6A      1800       ROR
1161  6A      1810       ROR
1162  6A      1820       ROR
1163  6A      1830       ROR
1164  2960    1840       AND 60
1166  850F    1850       STA *TO  +01
1168  A501    1860       LDA *YAT
116A  0A      1870       ASL
116B  0A      1880       ASL
116C  0A      1890       ASL
116D  AA      1900       TAX
116E  BD000C  1910       LDA HGRL,X
1171  850C    1920       STA *AT
1173  BD000D  1930       LDA HGRH,X
1176  291F    1940       AND 1F
1178  650D    1950       ADC *AT  +01
117A  850D    1960       STA *AT  +01
117C  A504    1970       LDA *YTO
117E  0A      1980       ASL
117F  0A      1990       ASL
1180  0A      2000       ASL
1181  AA      2010       TAX
1182  BD000C  2020       LDA HGRL,X
1185  850E    2030       STA *TO
1187  BD000D  2040       LDA HGRH,X
118A  291F    2050       AND 1F
118C  650F    2060       ADC *TO  +01
118E  850F    2070       STA *TO  +01
1190  A208    2080       LDX 08
1192  60      2090       RTS
              2100  :
1193  20C00C  2110  INIT JSR $0CC0
1196  A97F    2120       LDA 7F
1198  8D0160  2130       STA $6001
119B  8D0164  2140       STA $6401
119E  8D0168  2150       STA $6801
11A1  8D016C  2160       STA $6C01
11A4  8D0170  2170       STA $7001
11A7  8D0174  2180       STA $7401
11AA  8D0178  2190       STA $7801
11AD  8D017C  2200       STA $7C01
11B0  A900    2210       LDA 00
11B2  8508    2220       STA *YMAX
11B4  A901    2230       LDA 01
11B6  8507    2240       STA *XMAX
11B8  60      2250       RTS
              2260  :
              2270       .OR 1200
1200  98      2280  STOR TYA
1201  48      2290       PHA
1202  A503    2300       LDA *XTO
1204  8511    2310       STA *XIN
1206  A504    2320       LDA *YTO
1208  8512    2330       STA *YIN
120A  202C12  2340       JSR X40
120D  A513    2350       LDA *PROD+00
120F  850E    2360       STA *TO  +00
1211  18      2370       CLC
1212  A514    2380       LDA *PROD+01
1214  6900    2390       ADC 00
1216  850F    2400       STA *TO  +01
1218  A500    2410       LDA *XAT
121A  8511    2420       STA *XIN
121C  A501    2430       LDA *YAT
121E  8512    2440       STA *YIN
1220  202C12  2450       JSR X40
1223  A513    2460       LDA *PROD
1225  A000    2470       LDY 00
1227  910E    2480       STA (TO  ),Y
1229  68      2490       PLA
122A  A8      2500       TAY
122B  60      2510       RTS
122C  A512    2520  X40  LDA *YIN
122E  8513    2530       STA *PROD
1230  A900    2540       LDA 00
1232  8514    2550       STA *PROD+01
1234  0613    2560       ASL *PROD
1236  2614    2570       ROL *PROD+01
1238  0613    2580       ASL *PROD
123A  2614    2590       ROL *PROD+01
123C  0613    2600       ASL *PROD
123E  2614    2610       ROL *PROD+01
1240  A513    2620       LDA *PROD
1242  0613    2630       ASL *PROD
1244  2614    2640       ROL *PROD+01
1246  0613    2650       ASL *PROD
1248  2614    2660       ROL *PROD+01
124A  6513    2670       ADC *PROD
124C  8513    2680       STA *PROD
124E  A514    2690       LDA *PROD+01
1250  6900    2700       ADC 00
1252  8514    2710       STA *PROD+01
1254  A513    2720       LDA *PROD
1256  6511    2730       ADC *XIN
1258  8513    2740       STA *PROD
125A  A514    2750       LDA *PROD+01
125C  6900    2760       ADC 00
125E  8514    2770       STA *PROD+01
1260  60      2780       RTS
              2790  :
```

```
2800  XAT  .DS 0000
2810  YAT  .DS 0001
2820  ZAT  .DS 0002
2830  XTO  .DS 0003
2840  YTO  .DS 0004
2850  ZTO  .DS 0005
2860  SCOR .DS 0006
2870  XMAX .DS 0007
2880  YMAX .DS 0008
2890  XTMP .DS 0009
2900  YTMP .DS 000A
2910  BEST .DS 000B
2920  AT   .DS 000C
2930  TO   .DS 000E
2940  ERR  .DS 0010
2950  XIN  .DL 0011
2960  YIN  .DL 0012
2970  PROD .DL 0013
2980  HGRL .DS 0C00
2990  HGRH .DS 0D00
3000  BITS .DS 1000
3010  BELL .DS FF3A
3020  END  .EN
```

Listing 2.

```
0C00- 00 00 00 00 00 00 00 00
0C08- 80 80 80 80 80 80 80 80
0C10- 00 00 00 00 00 00 00 00
0C18- 80 80 80 80 80 80 80 80
0C20- 00 00 00 00 00 00 00 00
0C28- 80 80 80 80 80 80 80 80
0C30- 00 00 00 00 00 00 00 00
0C38- 80 80 80 80 80 80 80 80
0C40- 28 28 28 28 28 28 28 28
0C48- A8 A8 A8 A8 A8 A8 A8 A8
0C50- 28 28 28 28 28 28 28 28
0C58- A8 A8 A8 A8 A8 A8 A8 A8
0C60- 28 28 28 28 28 28 28 28
0C68- A8 A8 A8 A8 A8 A8 A8 A8
0C70- 28 28 28 28 28 28 28 28
0C78- A8 A8 A8 A8 A8 A8 A8 A8
0C80- 50 50 50 50 50 50 50 50
0C88- D0 D0 D0 D0 D0 D0 D0 D0
0C90- 50 50 50 50 50 50 50 50
0C98- D0 D0 D0 D0 D0 D0 D0 D0
0CA0- 50 50 50 50 50 50 50 50
0CA8- D0 D0 D0 D0 D0 D0 D0 D0
0CB0- 50 50 50 50 50 50 50 50
0CB8- D0 D0 D0 D0 D0 D0 D0 D0
```

```
0D00- 20 24 28 2C 30 34 38 3C
0D08- 20 24 28 2C 30 34 38 3C
0D10- 21 25 29 2D 31 35 39 3D
0D18- 21 25 29 2D 31 35 39 3D
0D20- 22 26 2A 2E 32 36 3A 3E
0D28- 22 26 2A 2E 32 36 3A 3E
0D30- 23 27 2B 2F 33 37 3B 3F
0D38- 23 27 2B 2F 33 37 3B 3F
0D40- 20 24 28 2C 30 34 38 3C
0D48- 20 24 28 2C 30 34 38 3C
0D50- 21 25 29 2D 31 35 39 3D
0D58- 21 25 29 2D 31 35 39 3D
0D60- 22 26 2A 2E 32 36 3A 3E
0D68- 22 26 2A 2E 32 36 3A 3E
0D70- 23 27 2B 2F 33 37 3B 3F
0D78- 23 27 2B 2F 33 37 3B 3F
0D80- 20 24 28 2C 30 34 38 3C
0D88- 20 24 28 2C 30 34 38 3C
0D90- 21 25 29 2D 31 35 39 3D
0D98- 21 25 29 2D 31 35 39 3D
0DA0- 22 26 2A 2E 32 36 3A 3E
0DA8- 22 26 2A 2E 32 36 3A 3E
0DB0- 23 27 2B 2F 33 37 3B 3F
0DB8- 23 27 2B 2F 33 37 3B 3F
```

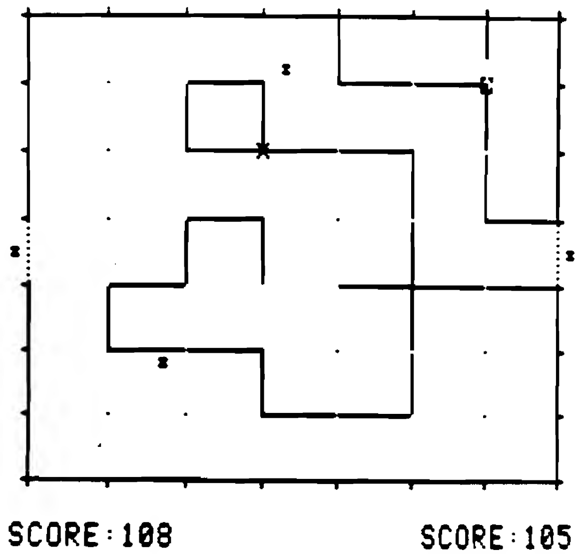
```
1000- 00 01 01 02 01 02 02 03
1008- 01 02 02 03 02 03 03 04
1010- 01 02 02 03 02 03 03 04
1018- 02 03 03 04 03 04 04 05
1020- 01 02 02 03 02 03 03 04
1028- 02 03 03 04 03 04 04 05
1030- 02 03 03 04 03 04 04 05
1038- 03 04 04 05 04 05 05 06
1040- 01 02 02 03 02 03 03 04
1048- 02 03 03 04 03 04 04 05
1050- 02 03 03 04 03 04 04 05
1058- 03 04 04 05 04 05 05 06
1060- 02 03 03 04 03 04 04 05
1068- 03 04 04 05 04 05 05 06
1070- 03 04 04 05 04 05 05 06
1078- 04 05 05 06 05 06 06 07
```

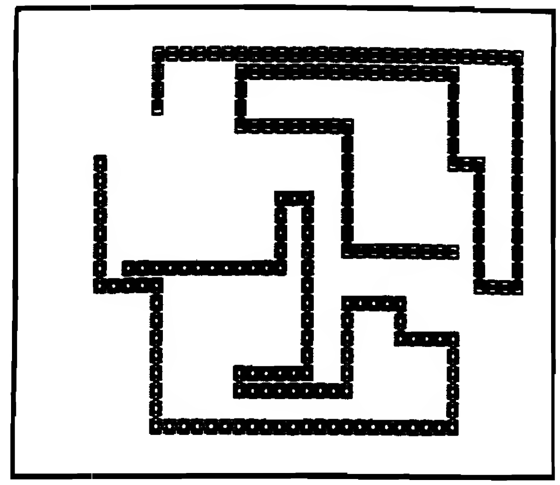
Listing 3.

```
  0 REM WRITTEN BY:  BOB BISHOP
 10 DIM A$(40)
 20 ANAL=11*256: SYN=ANAL+128: PRESS=
    4096+2*256+8*16
 30 FLAG=0: XFLAG=0
100 CALL -936:  POKE -16300,0:  POKE
    -16303,0
110 TAB 17:  PRINT "M E N U"
120 TAB 17:  PRINT "-------":  PRINT

130 PRINT :  PRINT "   L - LOAD PICTU
    RE FROM DISK"
140 PRINT :  PRINT "   A - ANALYZE PI
    CTURE INTO PIXELS"
150 PRINT :  PRINT "   S - SYNTHESIZE
     PICTURE FROM PIXELS"
160 PRINT :  PRINT "   1 - DISPLAY OR
    IGINAL PICTURE"
170 PRINT :  PRINT "   2 - DISPLAY SY
    NTHESIZED PICTURE"
```

```
180 PRINT : PRINT "   D - ISSUE DISK
     COMMANDS"
190 PRINT : PRINT "   X - SAVE COMPR
    ESSED PICTURE TO DISK"
195 VTAB 20: PRINT "SELECTION: "

200 REM READ KEYBOARD
210 CHAR= PEEK (-16384)
220 IF CHAR<128 THEN 210
230 POKE -16384+16,0
300 ID=0
310 IF CHAR= ASC("L") THEN ID=1

320 IF CHAR= ASC("A") THEN ID=2

330 IF CHAR= ASC("S") THEN ID=3

340 IF CHAR= ASC("1") THEN ID=4

350 IF CHAR= ASC("2") THEN ID=5

360 IF CHAR= ASC("D") THEN ID=6

370 IF CHAR= ASC("X") THEN ID=7

400 IF ID=0 THEN 100
500 GOTO 1000*ID
1000 VTAB 20: TAB 12: CALL -958:
      PRINT "LOAD PICTURE"
1005 POKE -16300,0: POKE -16303,
     0
1010 VTAB 22: INPUT "FILE NAME: "
     ,A$
1015 IF A$="" THEN 100
1020 VTAB 22: PRINT "BLOAD ";A$;
     ",A$2000,D1"
1050 GOTO 100
2000 VTAB 20: TAB 12: CALL -958:
      PRINT "ANALYZE PICTURE"
2005 POKE -16300,0: POKE -16303,
     0
2010 VTAB 22: INPUT "MAX ERRORS/PIXEL
     : ",MAXERR
2020 POKE 16,MAXERR: CALL ANAL
2025 FLAG=1:XFLAG=0:NUMBER=40* PEEK
     (8)+ PEEK (7)+1
2030 VTAB 22: PRINT "THERE ARE "
     ;NUMBER;" PIXELS WITH MAX ERROR
     = ";MAXERR
2035 POKE -16384+16,0
2040 IF PEEK (-16384)<128 THEN 2040

2050 GOTO 100
3000 VTAB 20: TAB 12: PRINT "SYNTHESI
     ZE PICTURE"
3005 POKE -16300,0: POKE -16303,
     0: VTAB 22: CALL -958
3010 FOR K=1 TO 500: NEXT K
3020 IF FLAG THEN 3050
3030 VTAB 22: PRINT "THERE ARE NO PIX
     ELS DEFINED YET!"
3040 GOTO 3060
3050 CALL SYN
3055 XFLAG=1
3060 POKE -16384+16,0
3070 IF PEEK (-16384)<128 THEN 3070

3080 IF PEEK (-16384)= ASC("1") THEN
     210
3085 IF PEEK (-16384)= ASC("2") THEN
     210
3090 GOTO 100
4000 POKE -16304,0: POKE -16302,
     0: POKE -16300,0: POKE -16297
     ,0
4050 GOTO 200
5000 POKE -16304,0: POKE -16302,
     0: POKE -16299,0: POKE -16297
     ,0
5050 GOTO 200
6000 VTAB 20: TAB 12: CALL -958:
      PRINT "DISK COMMAND"
6005 POKE -16300,0: POKE -16303,
     0
6010 VTAB 22: INPUT ":",A$
6015 IF A$="" THEN 100
6020 VTAB 22: TAB 2: PRINT "";A$

6030 PRINT : PRINT : PRINT
6040 GOTO 6010
7000 VTAB 20: TAB 12: CALL -958:
      PRINT "SAVE COMPRESSED PICTURE"

7005 POKE -16300,0: POKE -16303,
     0
7010 IF XFLAG THEN 7025
7015 VTAB 22: PRINT "NO PICTURE HAS B
     EEN SYNTHESIZED YET!"
7020 GOTO 7040
7025 IF NUMBER<=256 THEN 7060
7030 VTAB 22: PRINT "THERE ARE TOO MA
     NY (";NUMBER;") PIXELS"
7040 POKE -16384+16,0
7045 IF PEEK (-16384)<128 THEN 7045

7050 GOTO 100
7060 VTAB 22: INPUT "FILE NAME: "
     ,A$
7065 IF A$="" THEN 100
7070 CALL PRESS
7080 VTAB 22: PRINT "BSAVE ";A$;
     ",A$8000,L";960+2+8*NUMBER;
     ",D2"
7090 GOTO 100
```

Software for the Apple II





DYNAMAZE—a dazzling new real-time game. You move in a rectangular game grid, drawing or erasing walls to reflect balls into your goal (or to deflect them from your opponent's goal). Every ball in your goal is worth 100 points, but you lose a point for each unit of elapsed time and another point for each time unit you are moving. Control the speed with a game paddle: play as fast as ice hockey or as slowly and carefully as chess. Back up and replay any time you want to; it's a reversible game. By Don Stone. Integer Basic (plus machine language); 32 K; $9.95.

ULTRA BLOCKADE— the standard against which other versions have to be compared. Enjoy Blockade's superb combination of fast action (don't be the one who crashes) and strategy (the key is accessible open space—maximize yours while minimizing your opponent's). Play against another person or the computer. New high resolution graphics lets you see how you filled in an area—or use reversibility to review a game in slow motion (or at top speed, if that's your style). This is a game that you won't soon get bored with! By Don Stone. Integer Basic (plus machine language); 32 K; $9.95.

What is a REVERSIBLE GAME? You can stop the play at any point, back up and then do an "instant replay", analyzing your strategy. Or back up and resume the game at an earlier point, trying out a different strategy. Reversibility makes learning a challenging new game more fun. And helps you become a skilled player sooner.

WORLD OF ODYSSEY—a new adventure game utilizing the full power of Disk II, which enables the player to explore 353 rooms on 6 different levels full of dragons, dwarfs, orcs, goblins, gold and jewels. Applesoft II 48K; $19.95 includes diskette.

PERQUACKEY—an exciting vocabulary game which pits the player against the clock. The object of the game is to form words from a group of 10 letters which the computer chooses at random. The words must be 3 to 10 characters in length with no more than 5 words of any particular length. Each player has only 3 minutes per turn. The larger the words the higher the score. Applesoft II 16K; $9.95.

APPLESHIP—is a naval game in which two players enter their ships in respective oceans. Players take turns trying to blast their opponent's ships out of the water. The first player to destroy their opponent's ships may win the game. A great low-res graphics game. Applesoft II 32K; $14.95.

Available at your local computer store

Call or write for our free SOFTWARE CATALOG

Apple II is a registered trademark of Apple Computer, Inc.

DEALER INQUIRIES INVITED

POWERSOFT, INC.

P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

Programs Available on Diskette at $5.00 Additional

- Check or Money Order
- Include $1.00 for shipping and handling
- C.O.D. ($1.00 add'tl. charge)
- Master Charge and VISA orders accepted
- New Jersey residents add 5% sales tax

Assembly Language Applesoft Renumber

Alan D. Floeter
4333 N. 71 Street
Milwaukee, WI 53216

While there have been a number of programs published for renumbering APPLE BASIC, most have been written in BASIC and have therefore been slow. Here is a version written entirely in assembly language - very fast and very easy to use.

Chuck Carpenter gave a program in the May, 1979 issue of MICRO for renumbering Applesoft programs. Although this is probably adequate for most needs, there were still several drawbacks. Among these are the following:

1. User must make changes in BASIC instructions when the new line number has more digits than the original line number.
2. It is written in BASIC, so therefore, slower than a 6502 assembly language program.
3. The program will take up the same amount of memory, rather than reducing its size when it is possible.
4. User cannot specify only a portion of the program to be renumbered.
5. The program did not work for all types of IF-THEN statements.

Being a software person, I found it difficult to turn down the challenge to answer these deficiencies. The results of my efforts are contained in the following assembly language program.

To load the program, type in the hex numbers in the disassembled listing. This is written for a 32K or larger APPLE system. If you have a smaller system, you can go through the effort of relocating the program by hand. If you do relocate, be aware that the symbol table is stored at 7000 and continues as needed, using two bytes per line number. (A cassette version is available for $5 for any size system by contacting me. Make sure you state the amount of memory that you have. I will also give you a copy of this program at any other special memory location if you have a need for this.) Record from 6C00 to 6F9C.

To execute the renumberer, load your Applesoft program, Hit reset and load the binary executable renumber program. Type: 6C00G. You will now see a flashing cursor. Enter the line number in the Applesoft program where the renumbering will begin. Then enter the next statement number you do not want renumbered. Finally, enter the new line number to start with, followed by the increment between line numbers.

When the program is finished, (normally under 30 seconds,) type: 0G, and your program is renumbered. You can now record it, or continue developing it as normal.

An example of executing the program is as follows:

52	(Start at line number 52...)
512	(And stopping before line 512...)
60	(Renumber, start with line 60)
10	(And go in increments of 10)
0G	(And carry on!)

```
0C00-  A9 B0     LDA  #$B0
0C02-  8D E3 0F  STA  $0FE3
0C05-  A9 00     LDA  #$00
0C07-  85 FC     STA  $FC
0C09-  A9 02     LDA  #$02
0C0B-  85 FD     STA  $FD
0C0D-  20 EB 0D  JSR  $0DEB
0C10-  AD F4 0F  LDA  $0FF4
0C13-  8D E7 0F  STA  $0FE7
0C16-  AD F5 0F  LDA  $0FF5
0C19-  8D E8 0F  STA  $0FE8
0C1C-  20 EB 0D  JSR  $0DEB
0C1F-  AD F4 0F  LDA  $0FF4
0C22-  8D E5 0F  STA  $0FE5
0C25-  AD F5 0F  LDA  $0FF5
0C28-  8D E6 0F  STA  $0FE6
0C2B-  20 EB 0D  JSR  $0DEB
0C2E-  AD F4 0F  LDA  $0FF4
0C31-  8D FE 0F  STA  $0FFE
0C34-  AD F5 0F  LDA  $0FF5
0C37-  8D FF 0F  STA  $0FFF
0C3A-  20 EB 0D  JSR  $0DEB
0C3D-  AD F4 0F  LDA  $0FF4
0C40-  8D FC 0F  STA  $0FFC
0C43-  AD F5 0F  LDA  $0FF5
0C46-  8D FD 0F  STA  $0FFD
0C49-  A9 30     LDA  #$30
0C4B-  8D E3 0F  STA  $0FE3
0C4E-  A9 70     LDA  #$70
0C50-  85 FF     STA  $FF
0C52-  A9 00     LDA  #$00
0C54-  85 FE     STA  $FE
0C56-  AD FE 0F  LDA  $0FFE
0C59-  8D F6 0F  STA  $0FF6
0C5C-  A9 00     LDA  #$00
0C5E-  8D F8 0F  STA  $0FF8
0C61-  8D F9 0F  STA  $0FF9
0C64-  AD FF 0F  LDA  $0FFF
0C67-  8D F7 0F  STA  $0FF7
0C6A-  A5 07     LDA  $07
0C6C-  85 FC     STA  $FC
0C6E-  A5 08     LDA  $08
0C70-  85 FD     STA  $FD
0C72-  A0 00     LDY  #$00
0C74-  B1 FC     LDA  ($FC),Y
0C76-  48        PHA
0C77-  C8        INY
0C78-  B1 FC     LDA  ($FC),Y
0C7A-  48        PHA
0C7B-  F0 60     BEQ  $0CDD
0C7D-  C8        INY
0C7E-  A2 00     LDX  #$00
0C80-  B1 FC     LDA  ($FC),Y
0C82-  CD E7 0F  CMP  $0FE7
0C85-  C8        INY
0C86-  B1 FC     LDA  ($FC),Y
0C88-  ED E8 0F  SBC  $0FE8
0C8B-  90 47     BCC  $0CD4
0C8D-  88        DEY
0C8E-  B1 FC     LDA  ($FC),Y
0C90-  CD E5 0F  CMP  $0FE5
0C93-  C8        INY
0C94-  B1 FC     LDA  ($FC),Y
0C96-  ED E6 0F  SBC  $0FE6
0C99-  B0 42     BCS  $0CDD
0C9B-  88        DEY
0C9C-  B1 FC     LDA  ($FC),Y
0C9E-  81 FE     STA  ($FE,X)
0CA0-  C8        INY
0CA1-  E6 FE     INC  $FE
0CA3-  B1 FC     LDA  ($FC),Y
0CA5-  81 FE     STA  ($FE,X)
0CA7-  E6 FE     INC  $FE
0CA9-  D0 02     BNE  $0CAD
0CAB-  E6 FF     INC  $FF
0CAD-  EE F8 0F  INC  $0FF8
0CB0-  D0 03     BNE  $0CB5
0CB2-  EE F9 0F  INC  $0FF9
0CB5-  88        DEY
0CB6-  AD F6 0F  LDA  $0FF6
0CB9-  91 FC     STA  ($FC),Y
0CBB-  C8        INY
0CBC-  AD F7 0F  LDA  $0FF7
0CBF-  91 FC     STA  ($FC),Y
0CC1-  18        CLC
0CC2-  AD F6 0F  LDA  $0FF6
0CC5-  6D FC 0F  ADC  $0FFC
0CC8-  8D F6 0F  STA  $0FF6
0CCB-  AD F7 0F  LDA  $0FF7
0CCE-  6D FD 0F  ADC  $0FFD
0CD1-  8D F7 0F  STA  $0FF7
0CD4-  68        PLA
0CD5-  85 FD     STA  $FD
0CD7-  68        PLA
0CD8-  85 FC     STA  $FC
0CDA-  18        CLC
0CDB-  90 95     BCC  $0C72
0CDD-  68        PLA
0CDE-  68        PLA
0CDF-  A5 07     LDA  $07
0CE1-  85 FC     STA  $FC
0CE3-  A5 08     LDA  $08
0CE5-  85 FD     STA  $FD
0CE7-  A0 00     LDY  #$00
0CE9-  B1 FC     LDA  ($FC),Y
0CEB-  38        SEC
0CEC-  E5 FC     SBC  $FC
0CEE-  AA        TAX
0CEF-  CA        DEX
0CF0-  CA        DEX
0CF1-  CA        DEX
0CF2-  CA        DEX
0CF3-  A0 04     LDY  #$04
0CF5-  B1 FC     LDA  ($FC),Y
0CF7-  C9 B0     CMP  #$B0
0CF9-  F0 25     BEQ  $0D20
0CFB-  C9 AB     CMP  #$AB
0CFD-  F0 21     BEQ  $0D20
0CFF-  C9 C4     CMP  #$C4
0D01-  F0 1D     BEQ  $0D20
0D03-  C8        INY
0D04-  CA        DEX
0D05-  D0 EE     BNE  $0CF5
0D07-  98        TYA
0D08-  A0 01     LDY  #$01
0D0A-  18        CLC
0D0B-  65 FC     ADC  $FC
0D0D-  85 FC     STA  $FC
0D0F-  90 02     BCC  $0D13
0D11-  E6 FD     INC  $FD
0D13-  B1 FC     LDA  ($FC),Y
0D15-  D0 D0     BNE  $0CE7
0D17-  A5 B0     LDA  $B0
0D19-  85 0A     STA  $0A
0D1B-  A5 AF     LDA  $AF
0D1D-  85 09     STA  $09
0D1F-  60        RTS
0D20-  C8        INY
0D21-  CA        DEX
0D22-  F0 E3     BEQ  $0D07
0D24-  20 F0 0D  JSR  $0DF0
0D27-  AD EE 0F  LDA  $0FEE
0D2A-  F0 C9     BEQ  $0CF5
0D2C-  98        TYA
0D2D-  48        PHA
0D2E-  AD FE 0F  LDA  $0FFE
0D31-  8D F6 0F  STA  $0FF6
0D34-  AD FF 0F  LDA  $0FFF
0D37-  8D F7 0F  STA  $0FF7
0D3A-  A9 00     LDA  #$00
0D3C-  8D F2 0F  STA  $0FF2
0D3F-  8D F3 0F  STA  $0FF3
0D42-  A9 70     LDA  #$70
0D44-  85 FF     STA  $FF
0D46-  A9 00     LDA  #$00
0D48-  85 FE     STA  $FE
0D4A-  A0 00     LDY  #$00
0D4C-  B1 FE     LDA  ($FE),Y
0D4E-  C8        INY
0D4F-  CD F4 0F  CMP  $0FF4
0D52-  D0 07     BNE  $0D5B
0D54-  B1 FE     LDA  ($FE),Y
0D56-  CD F5 0F  CMP  $0FF5
0D59-  F0 47     BEQ  $0DA2
0D5B-  88        DEY
0D5C-  E6 FE     INC  $FE
0D5E-  E6 FE     INC  $FE
0D60-  D0 02     BNE  $0D64
0D62-  E6 FF     INC  $FF
0D64-  18        CLC
0D65-  AD F6 0F  LDA  $0FF6
0D68-  6D FC 0F  ADC  $0FFC
0D6B-  8D F6 0F  STA  $0FF6
0D6E-  AD F7 0F  LDA  $0FF7
0D71-  6D FD 0F  ADC  $0FFD
0D74-  8D F7 0F  STA  $0FF7
0D77-  EE F2 0F  INC  $0FF2
0D7A-  D0 03     BNE  $0D7F
0D7C-  EE F3 0F  INC  $0FF3
0D7F-  AD F2 0F  LDA  $0FF2
0D82-  CD F8 0F  CMP  $0FF8
0D85-  D0 C5     BNE  $0D4C
0D87-  AD F3 0F  LDA  $0FF3
0D8A-  CD F9 0F  CMP  $0FF9
0D8D-  D0 BD     BNE  $0D4C
0D8F-  68        PLA
0D90-  A8        TAY
0D91-  B1 FC     LDA  ($FC),Y
0D93-  C9 30     CMP  #$30
0D95-  90 4A     BCC  $0DE1
0D97-  C9 3A     CMP  #$3A
0D99-  B0 46     BCS  $0DE1
0D9B-  C8        INY
0D9C-  CA        DEX
0D9D-  D0 F2     BNE  $0D91
0D9F-  4C 07 0D  JMP  $0D07
0DA2-  20 32 0F  JSR  $0F32
0DA5-  68        PLA
0DA6-  A8        TAY
0DA7-  A9 00     LDA  #$00
0DA9-  8D F1 0F  STA  $0FF1
0DAC-  AD F0 0F  LDA  $0FF0
```

```
0DAF-   38          SEC
0DB0-   ED EE 0F    SBC   $0FEE
0DB3-   8D F0 0F    STA   $0FF0
0DB6-   10 05       BPL   $0DBD
0DB8-   A9 FF       LDA   #$FF
0DBA-   8D F1 0F    STA   $0FF1
0DBD-   F0 06       BEQ   $0DC5
0DBF-   20 03 0F    JSR   $0F03
0DC2-   20 51 0E    JSR   $0E51
0DC5-   8A          TXA
0DC6-   8D EF 0F    STA   $0FEF
0DC9-   A2 00       LDX   #$00
0DCB-   BD E9 0F    LDA   $0FE9,X
0DCE-   F0 06       BEQ   $0DD6
0DD0-   91 FC       STA   ($FC),Y
0DD2-   C8          INY
0DD3-   CE EF 0F    DEC   $0FEF
0DD6-   E8          INX
0DD7-   E0 05       CPX   #$05
0DD9-   D0 F0       BNE   $0DCB
0DDB-   AD EF 0F    LDA   $0FEF
0DDE-   AA          TAX
0DDF-   B1 FC       LDA   ($FC),Y
0DE1-   C9 2C       CMP   #$2C
0DE3-   D0 03       BNE   $0DE8
0DE5-   4C 20 0D    JMP   $0D20
0DE8-   4C F7 0C    JMP   $0CF7
0DEB-   20 0F FD    JSR   $FD0F
0DEE-   A0 00       LDY   #$00
0DF0-   98          TYA
0DF1-   48          PHA
0DF2-   A9 00       LDA   #$00
0DF4-   8D EE 0F    STA   $0FEE
0DF7-   8D F5 0F    STA   $0FF5
0DFA-   8D F4 0F    STA   $0FF4
0DFD-   B1 FC       LDA   ($FC),Y
0DFF-   38          SEC
0E00-   ED E3 0F    SBC   $0FE3
0E03-   90 42       BCC   $0E47
0E05-   C9 0A       CMP   #$0A
0E07-   B0 3E       BCS   $0E47
0E09-   EE EE 0F    INC   $0FEE
0E0C-   48          PHA
0E0D-   2E F4 0F    ROL   $0FF4
0E10-   2E F5 0F    ROL   $0FF5
0E13-   AD F5 0F    LDA   $0FF5
0E16-   48          PHA
0E17-   AD F4 0F    LDA   $0FF4
0E1A-   48          PHA
0E1B-   2E F4 0F    ROL   $0FF4
0E1E-   2E F5 0F    ROL   $0FF5
0E21-   2E F4 0F    ROL   $0FF4
0E24-   2E F5 0F    ROL   $0FF5
0E27-   68          PLA
0E28-   6D F4 0F    ADC   $0FF4
0E2B-   8D F4 0F    STA   $0FF4
0E2E-   68          PLA
0E2F-   6D F5 0F    ADC   $0FF5
0E32-   8D F5 0F    STA   $0FF5
0E35-   68          PLA
0E36-   6D F4 0F    ADC   $0FF4
0E39-   8D F4 0F    STA   $0FF4
0E3C-   A9 00       LDA   #$00
0E3E-   6D F5 0F    ADC   $0FF5
0E41-   8D F5 0F    STA   $0FF5
0E44-   C8          INY
0E45-   D0 B6       BNE   $0DFD
0E47-   68          PLA
0E48-   A8          TAY
0E49-   60          RTS
0E4A-   A5 B0       LDA   $B0
0E4C-   E9 00       SBC   #$00
0E4E-   4C 8D 0E    JMP   $0E8D
0E51-   98          TYA
0E52-   48          PHA
0E53-   8A          TXA
0E54-   48          PHA
0E55-   A5 AF       LDA   $AF
0E57-   85 FA       STA   $FA
0E59-   38          SEC
0E5A-   E5 FC       SBC   $FC
0E5C-   8D FA 0F    STA   $0FFA
0E5F-   A5 B0       LDA   $B0
0E61-   85 FB       STA   $FB
0E63-   E5 FD       SBC   $FD
0E65-   8D FB 0F    STA   $0FFB
0E68-   8C EF 0F    STY   $0FEF
0E6B-   AD FA 0F    LDA   $0FFA
0E6E-   38          SEC
0E6F-   ED EF 0F    SBC   $0FEF
0E72-   8D FA 0F    STA   $0FFA
0E75-   90 03       BCC   $0E7A
0E77-   EE FB 0F    INC   $0FFB
0E7A-   18          CLC
0E7B-   A5 AF       LDA   $AF
0E7D-   6D F0 0F    ADC   $0FF0
0E80-   85 AF       STA   $AF
0E82-   85 F8       STA   $F8
0E84-   AD F0 0F    LDA   $0FF0
0E87-   30 C1       BMI   $0E4A
0E89-   A5 B0       LDA   $B0
0E8B-   69 00       ADC   #$00
0E8D-   85 B0       STA   $B0
0E8F-   85 F9       STA   $F9
0E91-   AD F0 0F    LDA   $0FF0
0E94-   30 35       BMI   $0ECB
0E96-   A0 00       LDY   #$00
0E98-   B1 FA       LDA   ($FA),Y
0E9A-   91 F8       STA   ($F8),Y
0E9C-   38          SEC
0E9D-   A5 FA       LDA   $FA
0E9F-   E9 01       SBC   #$01
0EA1-   85 FA       STA   $FA
0EA3-   B0 02       BCS   $0EA7
0EA5-   C6 FB       DEC   $FB
0EA7-   38          SEC
0EA8-   A5 F8       LDA   $F8
0EAA-   E9 01       SBC   #$01
0EAC-   85 F8       STA   $F8
0EAE-   B0 02       BCS   $0EB2
0EB0-   C6 F9       DEC   $F9
0EB2-   38          SEC
0EB3-   AD FA 0F    LDA   $0FFA
0EB6-   E9 01       SBC   #$01
0EB8-   8D FA 0F    STA   $0FFA
0EBB-   B0 DB       BCS   $0E98
0EBD-   CE FB 0F    DEC   $0FFB
0EC0-   D0 D6       BNE   $0E98
0EC2-   18          CLC
0EC3-   68          PLA
0EC4-   6D F0 0F    ADC   $0FF0
0EC7-   AA          TAX
0EC8-   68          PLA
0EC9-   A8          TAY
0ECA-   60          RTS
0ECB-   A5 FC       LDA   $FC
0ECD-   85 F8       STA   $F8
0ECF-   A5 FD       LDA   $FD
0ED1-   85 F9       STA   $F9
0ED3-   AD F0 0F    LDA   $0FF0
0ED6-   49 FF       EOR   #$FF
0ED8-   18          CLC
0ED9-   69 01       ADC   #$01
0EDB-   18          CLC
0EDC-   65 FC       ADC   $FC
0EDE-   85 FA       STA   $FA
0EE0-   A5 FD       LDA   $FD
0EE2-   69 00       ADC   #$00
0EE4-   85 FB       STA   $FB
0EE6-   B1 FA       LDA   ($FA),Y
0EE8-   91 F8       STA   ($F8),Y
0EEA-   C8          INY
0EEB-   D0 04       BNE   $0EF1
0EED-   E6 FB       INC   $FB
0EEF-   E6 F9       INC   $F9
0EF1-   38          SEC
0EF2-   AD FA 0F    LDA   $0FFA
0EF5-   E9 01       SBC   #$01
0EF7-   8D FA 0F    STA   $0FFA
0EFA-   B0 EA       BCS   $0EE6
0EFC-   CE FB 0F    DEC   $0FFB
0EFF-   D0 E5       BNE   $0EE6
0F01-   F0 BF       BEQ   $0EC2
0F03-   98          TYA
0F04-   48          PHA
0F05-   A5 FC       LDA   $FC
0F07-   85 FA       STA   $FA
0F09-   A5 FD       LDA   $FD
0F0B-   85 FB       STA   $FB
0F0D-   A0 01       LDY   #$01
0F0F-   B1 FA       LDA   ($FA),Y
0F11-   F0 1C       BEQ   $0F2F
0F13-   48          PHA
0F14-   88          DEY
0F15-   B1 FA       LDA   ($FA),Y
0F17-   48          PHA
0F18-   18          CLC
0F19-   6D F0 0F    ADC   $0FF0
0F1C-   91 FA       STA   ($FA),Y
0F1E-   C8          INY
0F1F-   B1 FA       LDA   ($FA),Y
0F21-   6D F1 0F    ADC   $0FF1
0F24-   91 FA       STA   ($FA),Y
0F26-   68          PLA
0F27-   85 FA       STA   $FA
0F29-   68          PLA
0F2A-   85 FB       STA   $FB
0F2C-   18          CLC
0F2D-   90 E0       BCC   $0F0F
0F2F-   68          PLA
0F30-   A8          TAY
0F31-   60          RTS
0F32-   8A          TXA
0F33-   48          PHA
0F34-   A9 30       LDA   #$30
0F36-   A2 04       LDX   #$04
0F38-   9D E9 0F    STA   $0FE9,X
0F3B-   CA          DEX
0F3C-   10 FA       BPL   $0F38
0F3E-   AD F7 0F    LDA   $0FF7
0F41-   85 FB       STA   $FB
0F43-   AD F6 0F    LDA   $0FF6
0F46-   85 FA       STA   $FA
0F48-   A0 00       LDY   #$00
0F4A-   E8          INX
0F4B-   B9 92 0F    LDA   $0F92,Y
0F4E-   F0 1F       BEQ   $0F6F
0F50-   38          SEC
0F51-   A5 FA       LDA   $FA
0F53-   F9 92 0F    SBC   $0F92,Y
0F56-   48          PHA
0F57-   C8          INY
0F58-   A5 FB       LDA   $FB
0F5A-   F9 92 0F    SBC   $0F92,Y
0F5D-   90 0B       BCC   $0F6A
0F5F-   85 FB       STA   $FB
0F61-   68          PLA
0F62-   85 FA       STA   $FA
0F64-   88          DEY
0F65-   FE E9 0F    INC   $0FE9,X
0F68-   D0 E6       BNE   $0F50
0F6A-   C8          INY
0F6B-   68          PLA
0F6C-   E8          INX
0F6D-   D0 DC       BNE   $0F4B
0F6F-   A2 00       LDX   #$00
0F71-   BD E9 0F    LDA   $0FE9,X
0F74-   C9 30       CMP   #$30
0F76-   D0 0A       BNE   $0F82
0F78-   A9 00       LDA   #$00
0F7A-   9D E9 0F    STA   $0FE9,X
0F7D-   E8          INX
0F7E-   E0 04       CPX   #$04
0F80-   D0 EF       BNE   $0F71
0F82-   8A          TXA
0F83-   8D F0 0F    STA   $0FF0
0F86-   A9 05       LDA   #$05
0F88-   38          SEC
0F89-   ED F0 0F    SBC   $0FF0
0F8C-   8D F0 0F    STA   $0FF0
0F8F-   68          PLA
0F90-   AA          TAX
0F91-   60          RTS
0F92-   10 27       BPL   $0FBB
0F94-   E8          INX
0F95-   03          ???
0F96-   64          ???
0F97-   00          BRK
0F98-   0A          ASL
0F99-   00          BRK
0F9A-   01 00       ORA   ($00,X)
0F9C-   00          BRK
```

Performing Math Functions in Machine Language

Alfred J. Bruey
201 S. Grinnell Street
Jackson, MI 49203

If you are afraid to try doing mathematical functions in assembly language, then this article may help you get started.

Since addition, subtraction and shifting are the only arithmetic functions available in machine language for most small computers, it becomes necessary to find methods to perform other mathematical operations using addition, subtraction, and shifts in combination with other commands available on the programmer's microprocessor.

Multiplication is an example of an operation that is commonly performed in this way. Let's look at a particular example. Suppose we want to multiply 187 by 345. It is obvious that we can clear a register and add 187 a total of 345 times to arrive at the answer, but we soon discover that it is more efficient to perform the same function by combining additions with shifts.

Using the shift command, we would add 3 187s, then shift left, then add 4 187s, the shift left, then add 5 187s to arrive at the final product. Thus, we have replaced 345 additions with 12 additions and 2 shifts. In the same way, repeated subtractions may be combined with shifts to implement a division algorithm.

Division and multiplication algorithms are often described in the programming manuals that come with a computer. A programmer soon needs other mathematical functions and must find a way to perform them with a limited instruction set and limited computer memory. If the functions become too complicated, one must add memory or go to a higher level language, such as BASIC.

The purpose of this article is to demonstrate the power of the lowly addition and subtraction commands by developing an algorithm for extracting the square root of a number. The algorithm is described and a flow chart is presented along with a 6502 listing for the KIM-1.

The square root algorithm to be presented here is based on the equation:

$$\sum_{k=1}^{n} (2k-1) = n^2 \text{; n an integer greater than 0}$$

which says, in English, that the sum of the first n odd integers is equal to the square of n. For example:

$$1+3+5+7+9=25=5^2$$

That is, the sum of the first 5 odd integers is equal to 5^2, or 25. This equation is easily proven true for all positive integers by mathematical induction.

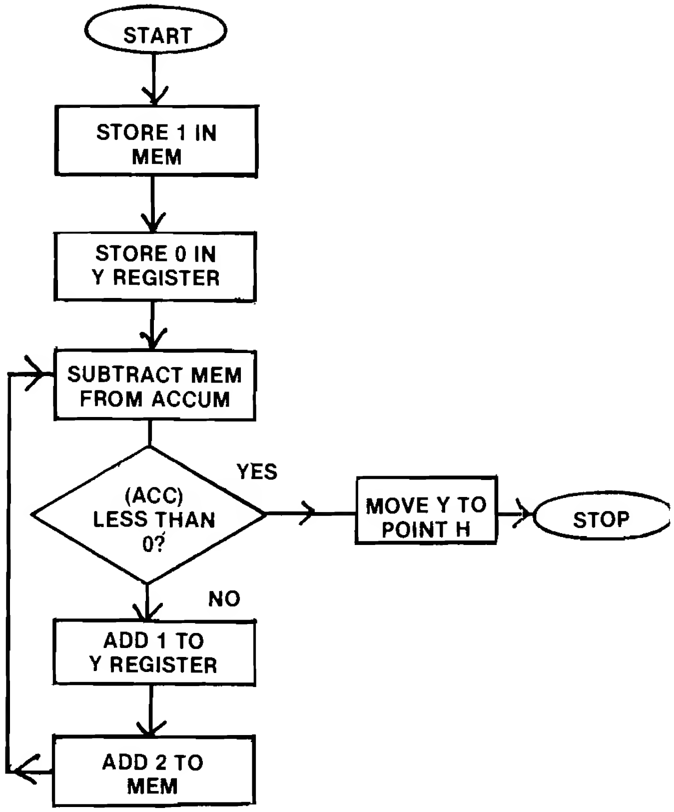
The method implemented here is to subtract first 1, then 3, then 5, and so on, from the number whose square root is desired. The number of subtractions, less 1, that it takes to reduce the original number to a nonzero negative number is the square root. For example, if X = 25:

$$25-1=24;\ 24-3=21;\ 21-5=16$$
$$16-7=9;\ 9-9=0;\ 0-11=11$$

Since it took 6 subtractions to reduce 25 to a number less than 0, the answer is 6 − 1 = 5. Notice that this method gives only the integer part of the answer, so if X had been any value from 25 to 35, you would have arrived at the same answer. Remember—when you take the square root of a number, your answer has only about half as many significant digits as the number.

The original value (NUM) is placed in the accumulator. The answer will be in the Y register and also displayed on the KIM's seven segment LEDs (POINTH). Notice that the algorithm as described below will not handle very large numbers. To use this for practical problems, it will have to be extended to multiple precision.

The coding to implement the routine is given below. While the addresses are given for the KIM-1, a few address changes should make it possible to implement this routine on any other 6502 based system. The number you want the square root of goes in location 0001, then set the address to 0000 and GO. The answer will be displayed in POINTH, the left two LEDs of the KIM display. The code given is probably not optimum—I am a relative newcomer to machine language coding. If you come with an improved version of this routine, I'd appreciate receiving a copy of it. The example shown is set to take the square root of $10.



Flowchart

```
0000:
0010:                    SQUARE ROOT ROUTINE
0020:
0030:                    ALFRED J. BRUEY
0040:
0050: 0000                      ORG   $0000
0060:
0070: 0000               MEM    *     $001A
0080: 0000               POINTH *     $00FB
0090: 0000               START  *     $1C4F
0100:
0110: 0000 A9 10         BEGIN  LDAIM $10
0120: 0002 A0 01                LDYIM $01
0130: 0004 86 1A                STY   MEM
0140: 0006 A0 00                LDYIM $00
0150:
0160: 0008 38            LOOP   SEC
0170: 0009 E5 1A                SBC   MEM
0180: 000B 30 08                BMI   STOP
0190: 000D C8                   INY
0200: 000E E6 1A                INC   MEM
0210: 0010 E6 1A                INC   MEM
0220: 0012 4C 08 00             JMP   LOOP
0230:
0240: 0015 84 FB         STOP   STY   POINTH
0250: 0017 4C 4F 1C             JMP   START
0260:
ID=
```

apple computer®

Apple II Reference Manual $10.00
Apple Soft Manual 10.00
Programmer's Guide (Computer Station) 5.95
Apple II Monitor Peeled 9.95
Software Directory for Apple
- Business, Finance & Utility 4.95
- Games, Demo, Utility 4.95

Best of Contact '78 .... 2.50
Programming in PASCAL (Grogono) 9.90

INTERFACE CARDS

Prototyping/Hobby Card	$ 24.00
Parallel Printer Interface Card	180.00
Communications Card & DB25 Connector Cable	225.00
High-Speed Serial Interface Card	195.00
Language System with Pascal (48K RAM & Disk II Required)	495.00
Applesoft II Firmware Card	200.00
16 Input Analog Card	295 00

ACCESSORIES

Disk II—Drive Only	495.00
Disk II—Drive & Controller (32K Min. RAM Recommended)	595.00
Vinyl Carrying Case	30.00
Tape Recorder	40.00
Programmers Aid No. 1 Firmware (For Use with Integer BASIC)	50.00
Clock/Calendar Card	199.00
Auto—Start ROM Package (For Apple II Only)	65.00
Digikitizer Pad by Talos (Kitform)	499.00

High Resolution Light Pen	199.00
Micromodem (D.C. Hayes)	379.00
12" B/W Leebex Monitor	149.00
Cable from Monitor to Apple II	9.95
13" Color TV Compatible with Apple II	290.00
Sup-R-Modulator (RF)	25.00

SOFTWARE FOR APPLE II

PASCAL from Programma	49.95
FORTH	49.95
LISP—from Apple Software Bk No. 3	N/C
LISA—Interactive disk assembler	34.95
WHATSIT—Excellent conversational data base manager	32K 100.00 48K 125.00
SARGON-Champ of 2nd West Coast Computer Faire	19.95
APPLE PIE—Excellent text editor	24.95
FORTE—Music editor in hires	19.95
FASTGAMMON—Excellent backgammon game with graphics	Tape 20.00 Disk 25.00
APPLE 21—Excellent blackjack game	9.95
BRIDGE CHALLENGER—Computer bridge	14.95

FINANCIAL MANAGEMENT SYSTEM
- Accounts Payable
- Accounts Receivable
- Inventory Control
- $200 Each Package
- Ledger Processing
- Payroll
- $800 Complete
- $10 for Manual

PRINTER SPECIALS FOR APPLE AND PET

TRENDCOM 100 with interface for Apple or PET	450.00
LITE PEN used with TV or monitor screen	34 95
ALF Music Synthesizer Boards	265.00
Supertalker	279 00
Anadex DP-8000 with tracter 8" paper width and interface to Apple	1050.00
Centronics 779-2 for Apple II with parallel interface	1245.00

SOFTWARE (Send For complete Software Catalog $1.00)

Dow Jones Portfolio Evaluator/Stock Quote Reporter Disk	50.00
Microchess 2.0 Chess Disk	25 00
Disk Utility Pack with DOS 3.2	25.00
The Controller (General Business System)	625.00
Apple Post (Mailing List System)	49.95
Bowling Program Diskette-	15.00
The Cashier (Retail Store Management	250.00
Checkbook Cassette	20.00
Applesoft II Language & Demo Cassette	20 00
RAM Test Tape with Manual	7 50
Finance 1-2 Cassette Package	25 00
Datamover/Telepong Cassette (Com. Card & Modem Req'd)	7.50
Microchess 2.0 Chess Tape	20.00
Bowling Program Tape	15 00
Pascal with Language System (48K RAM & Disk II Required)	495 00

New for Apple Computer Owners at Low ComputerWorld Prices

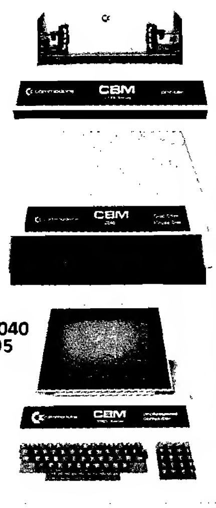
8" Disk Drives with housing **$1295.00** for single drive **$1895.00** for dual drive

apple computer

Joystick for Apple II **Only $39.95 each.**
ComputerWorld's Complete Library of 600 Apple II programs **Only $60.00**

A PROFESSIONAL BUSINESS SYSTEM

commodore CBM™



CMB 2022 $995

CBM 2040 $1295

2001-16B $995

2001-8 .... $795
2001-16N $995
2001-32N $1295
2001-32B $1295
External Cassette $95
PET to IEEE Cable $39.95
CBM 2023 Printer $849
IEEE to IEEE Cable $49.95

PET™

Commodore PET Service Kit $30.00
Beeper—Tells when tape is Loaded 24.95
Petunia—Play music with PET 29.95
Video Buffer—Attach another display 29.95
Combo—Petunia and Video Buffer 49.95

SOFTWARE FOR PET

Mirrors and Lenses	19.95
The States	14.95
Real Estate 1 & 2	59.95
Momentum and Energy	19.95
Projectile Motion	19.95
Mortgage	14.95
Dow Jones	7.95
Petunia Player Sftwr	14.95
Checkers and Baccarat	7.95
Chess	19.95
Series and Parallel	
Circuit Analysis	19.95
Home Accounting	9.95
BASIC Math	29.95
Game Playing with BASIC Vol. I, II, III	9.95 each

Join Now

Become a member of ComputerWorld's RAYGAMCO Computer Discount Club.
By being a RAYGAMCO Member you receive substantial discounts on every item you purchase, including all hardware, software, accessories, even books and paper! You will also receive a monthly newsletter with all the latest available for your particular computer system, and much, much more — exclusive to RAYGAMCO Members only!

Here's how to join.
Nothing to buy. Simply complete the self-addressed postcard in this magazine with name, address, and your computer system. You'll receive your personalized RAYGAMCO Computer Discount Club Membership Card in the mail with information on how to use your card when ordering for big savings!

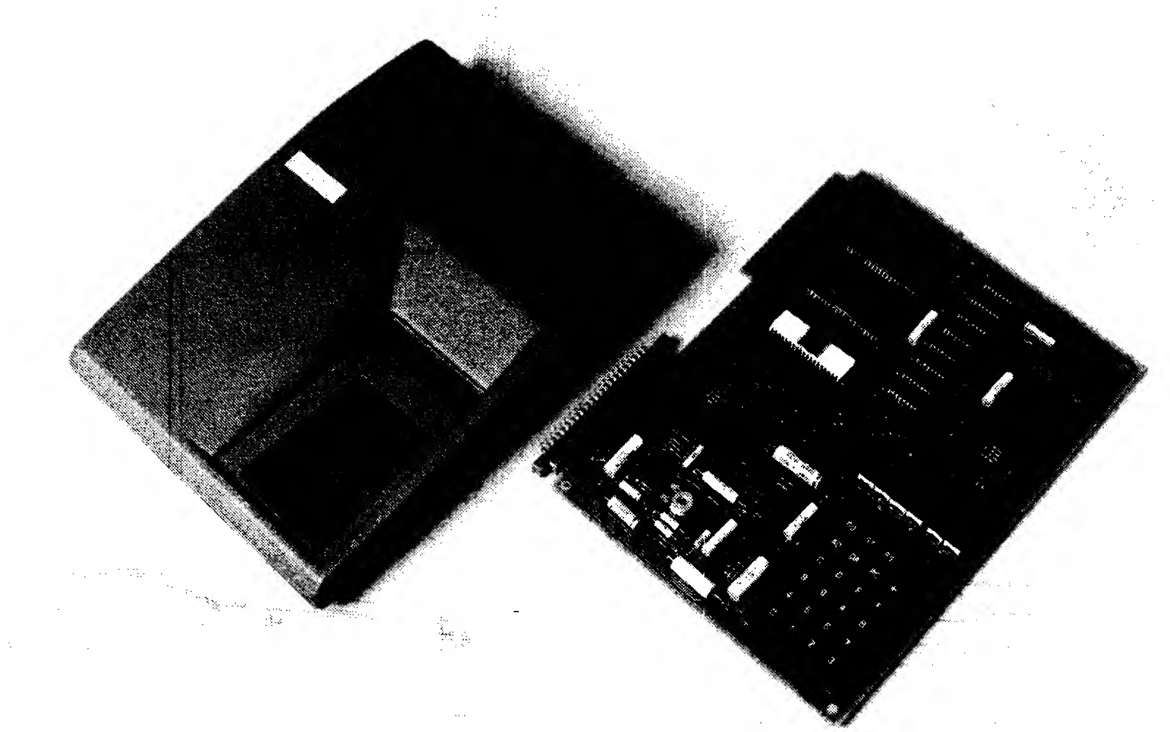
Charter RAYGAMCO Members' Special. Join now and receive 20% OFF of any purchase.*

*20% offer expires December 24, 1979. Offer is valid for RAYGAMCO Members only.

ComputerWorld

A RAYGAM COMPANY

6791 Westminster Ave., Westminster, CA 92683 (714) 891-2587



TSAR: A Time Sharing Administrative Routine for the KIM-1

If you think the KIM-1 is too small to do interesting jobs, then consider this program. TSAR is a super monitor which supports time-sharing, opening the door to a wide variety of new capabilities. The techniques can easily be translated for use on other computers.

Philip K. Hooper
3 Washington Street
Northfield, VT 05663

The program presented here takes over supervisory control of the KIM-1, demoting the KIM monitor to the role of "just another program" sharing execution time with a list of user programs. The monitor, with its display and keypad, remains available while user programs are running, permitting true "front panel" operation; examination and even alteration of memory during program execution. There is provision for inserting breakpoints into a program while it is running, as well as a TSAR-compatible breakpoint servicing routine. Although the system as presented is configured for six programs, in addition to the monitor and TSAR itself, it is easily expanded to provide supervision for as many user programs as memory and stack requirements permit.

Introduction

Not long ago, if anyone had suggested to me that I should write a timesharing system for my KIM-l, I would have objected on two counts: first, that it would be pointless with such a small, single-user system; second, that it would be far too complicated to design, implement, and operate. I would have been wrong on both counts.

I had been working on a design problem — the problem of providing a perfectly transparent operating environment for my TVT-6 video board. This inexpensive and very versatile board draws its timing signals directly from the address bus of the MPU and can not function normally without the full, undivided attention of KIM, making its use along with another program rather awkward to orchestrate.

When I had finally tamed the microseconds and sync pulses and had my transparent display operational, I loaded my LIFE program and settled back, regarding the result with satisfaction and noting how cooperative it all seemed, with the display driver and LIFE program sharing the time of the MPU. And then it hit me! This was already a timesharing system. Moreover, leaving out the TVT-6 would let me streamline the system and also extend it to the supervision of many "simultaneous" programs.

Before explaining the operation of the system, let me note resources the system requires as well as some of the features it offers. Its needs are few: an unexpanded KIM-1 provides sufficient memory for overseeing the operation of thirty-some programs; the supervisory routine, TSAR, resides in forty-four bytes of page twenty-three; a special, and optional, breakpoint routine occupies another fifteen bytes on that page; fifteen page zero locations are required for storing system variables under a six-user-program configuration (with two additional bytes needed for each program over six); and page one is distributed as stack space for the various programs.

The only hardware expansion needed is a wire, or possibly a switch, allowing the interval timer to send an interrupt to the MPU. (See Figure 1.) A speaker, connected as in the *Kim User Manual,* can provide dramatic examples of the system's use, but is certainly not essential to its operation.

The most useful aspect of the system is, in my opinion, provision for a full hex front panel. Under the KIM monitor, the keypad and display are used almost exclusively to enter and initiate programs. Though individual programs may use them for special purposes, they generally remain idle during program execution. Under TSAR, however, the monitor is timeshared and becomes a monitor in the full sense of the word, remaining active while other programs are executing. This permits the on-line examination (and even alteration) of any memory location, so that one can, for instance: watch a counter as it approaches zero, alter the value of a byte of data to determine its effect on the program, or even change an instruction opcode, all while the program is running! Essentially it brings full interaction to KIM-1, letting the user and running programs interact through the services of the monitor.

The cost of this continuous monitoring is time—the user programs run more slowly when timeshared—but there are occasions, as during certain program development stages, where this can be an advantage. By using this system with five dummy programs having large time slices, we produce an interactive slow-stepper. By letting the programs modify each other's time slices (an unnatural activity recommended only for producing unpredictable results) we can create an enormous variety of unusual timing patterns.

The Timesharing Procedure

The 6502 provides ready access to and manipulation of the stack pointer, and this in turn permits the realization of a fairly simple timesharing procedure. The programs to be run are placed in a queue and are activated and recalled by TSAR as it cycles through the queue. So TSAR can keep adequate track of these programs, each has its own stack area, stack pointer, and time slice determining how long the program will be active when its turn comes. The user selects the stack areas from page one, while the corresponding stack pointers and the time slices are kept in two page zero lists, STAX and TIMES respectively. The index number (position in the queue) of the program currently executing is stored in location INDEX. Figure 2 illustrates this procedure.

Assume that one of these programs, say P2, is running. Under TSAR, the interval timer will also be running, and armed, loaded initially with the time slice for P2 from TIMES+2. At "time out," TSAR will be re-entered at location 1780, via the NMI vector, and after disabling further interrupts TSAR will save registers A, X, and Y on the stack reserved for P2. P and PC have already been saved there, as part of the interrupt response of the 6502. TSAR will then

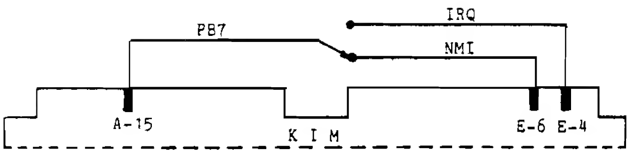


Figure 1: Enabling the timer interrupts. A SPDT switch connecting PB7 to either NMI or IRQ permits the fullest realization of TSAR's capability. The switch setting should not be changed without first setting both interrupt vectors to point to the monitor.

replace the stack pointer value for P2 in STAX + 2, the second position of the STAX list. This procedure is illustrated in Figure 3.

Disabling further interrupts at this time has no effect on the minimal configuration but is adviseable if there are other devices that could pull IRQ low connected to the system, to inhibit interrupts from them during operation of the supervisory routine.

Program 2 now remains idle until INDEX again assumes the value of 2. Before that occurs, TSAR will have looked at six other INDEX values; activated and later recalled those programs currently enabled—those with non-zero time slices; kept the monitor enabled; and maintained STAX as needed. At any rate, when INDEX = 2 does recur, the time slice for P2 will be brought in from TIMES + 2 and examined by TSAR. If this is zero, P2 is disabled and will be passed by. Otherwise, this time slice will be written to the timer, to initiate another time period during which P2 may execute.

Next, the stack pointer specific to P2 will be brought back from STAX + 2 and used to access P2's private stack, from which the saved values of the Y, X, and A registers will be pulled. Finally, an RTI will draw the status register and program counter from this same stack, and P2 will be off and running again, from the very place at which it was interrupted. If no other program interferes with its storage areas, P2 will function as though it were the only program in the KIM, although a bit more slowly.

In a small system like this, without software-initiated memory protect or disk-based page swapping, any unwanted interaction between programs must be prevented by the programmer. This is managed through carefully planned memory allocation and through the use of stack storage to make any shared routines fully reentrant—using different storage areas (stacks) depending on which program is using the routine. Therefore, whenever the monitor is included as an enabled program, the monitor subroutines which use RAM temporary storage and those which serve the monitor's keypad and display routines should not be called by a user program. The results would be unpredictable and would probably prevent interactive use of the monitor.

This sequence of execution, interruption, dormancy, reactivation is followed by all programs on the queue, including the monitor. Depending on its time slice, each enabled program receives from 64 to 16320 microseconds of execution time, minus TSAR's overhead, when its turn arrives, while those disabled by a null time slice are simply passed over. With six programs enabled in addition to the monitor, TSAR exacts roughly 80 microseconds to process each interrupt, and each disabled program increases this by about 20 microseconds to a maximum of 200 with the monitor alone enabled. The more work the system has to do, the more efficient it becomes! Of the above times, 30 microseconds is taken from the time slice of the program being reactivated, so that a time slice of 01, representing a single sixty-four microsecond portion, will actually provide thirty-four microseconds of execution time for the program, each time around.

A time slice range of from 1024 to 261,120 microseconds can be installed by replacing the value of the byte at 17A0 with "0F", which starts the timer's counter in the divide-by-1024 mode instead of the divide-by-64 mode. Although this reduces the relative time penalty charged by TSAR, it also degrades the response of the monitor somewhat.

Oddly enough, this is an example of one of the peculiar charms of the TSAR system. Some of the aggrevations that TSAR introduces—monitor response annoyingly slow at times, startup routine hard to remember, recovery from a crash a major undertaking—all of these provide the peculiar sensation that one is working on some sort of monster system and not just a KIM-1 with 1K of memory and a 50-odd byte timesharing supervisor.

```
                    *
                    * TSAR: BY PHILIP K. HOOPER
                    * MODIFIED 7-19-79 BY MICRO STAFF
                    *
17BC                COUNT  *      $0000
17BC                X      *      $0300

17BC                INDEX  *      $00E0
17BC                TIMES  *      $00E1
17BC                STAX   *      $00E8
17BC                POINTL *      $00FA

17BC                TIMER  *      $170E
17BC                DELAY  *      $1ED4
17BC                INCPT  *      $1F63
1780                       ORG    $1780
1780 78             ENTER  SEI           PLEASE DO NOT DISTURB
1781 48                    PHA           PLACE
1782 8A                    TXA           ALL
1783 48                    PHA           REGISTERS
1784 98                    TYA           ON THE
1785 48                    PHA           STACK
1786 BA                    TSX           GET CURRENT STACK POINTER FROM MPU
1787 A4 E0                 LDY    INDEX  NOW, WHICH PROGRAM IS RECALLED?
1789 96 E8                 STXZY  STAX   STASH STACK POINTER FOR INTERRUPTED PROGRAM
178B A5 E1                 LDA    TIMES  EXAMINE TIMING CONSTANT OF MONITOR
178D D0 02                 BNE    INDEC  IF ZERO, MONITOR DISABLED - MUST RESTORE
178F C6 E1                 DEC    TIMES  RESET MONITOR TIME SLICE
1791 C6 E0          INDEC  DEC    INDEX  DECREMENT INDEX TO NEXT PROGRAM
1793 10 04                 BPL    TINDX  POSITIVE DENOTES VALID INDEX
1795 A0 06                 LDYIM  $06    OTHERWISE, RESET INDEX TO POINT TO THE
1797 84 E0                 STY    INDEX  GREATEST PROGRAM QUEUE INDEX
1799 A6 E0          TINDX  LDX    INDEX  TENTATIVE INDEX OF NEXT PROGRAM
179B B5 E1                 LDAX   TIMES  FETCH PROGRAM TIME SLICE
179D F0 F2                 BEQ    INDEC  IF ZERO, PROGRAM DISABLED - PICK ANOTHER
179F 8D 0E 17              STA    TIMER  DEPOSIT TIMING INTERVAL IN COUNTER - ENABLE INT
17A2 B5 E8                 LDAX   STAX   FETCH STACK POINTER FOR REACTIVATED PROGRAM
17A4 AA                    TAX           PUT STACK POINTER IN X REGISTER
17A5 9A                    TXS           AND DEPOSIT STACK POINTER INTO MPU
17A6 68                    PLA           BRING
17A7 A8                    TAY           IN
17A8 68                    PLA           REGISTERS
17A9 AA                    TAX           OFF
17AA 68                    PLA           STACK
17AB 40             LEAVE  RTI           RETURN TO THE PROGRAM
```

The code for this procedure is presented in the listing. Note that the sections of code for the normal (interrupt to 1C00) entry and normal ("GO" -1DC8) exit for the ROM monitor are closely related to the entry and exit code for TSAR. Both involve storing, and later retrieving, the contents of the PCH, PCL, P, A, Y, X, and SP registers, which together completely specify the internal state of the MPU. However, while the monitor stores these values in fixed, page zero locations, TSAR places them in a user stack reserved for the particular program which has just been recalled.

Using the monitor as the operating system, the user can alter these zero page locations holding register values, making it possible to exit from the monitor to a different program, with a different set of operating parameters, than the program that was running before. TSAR does this automatically, by pulling the register values from a different stack, the one corresponding to the program about to be activated, rather than from the stack for the program which was just recalled.

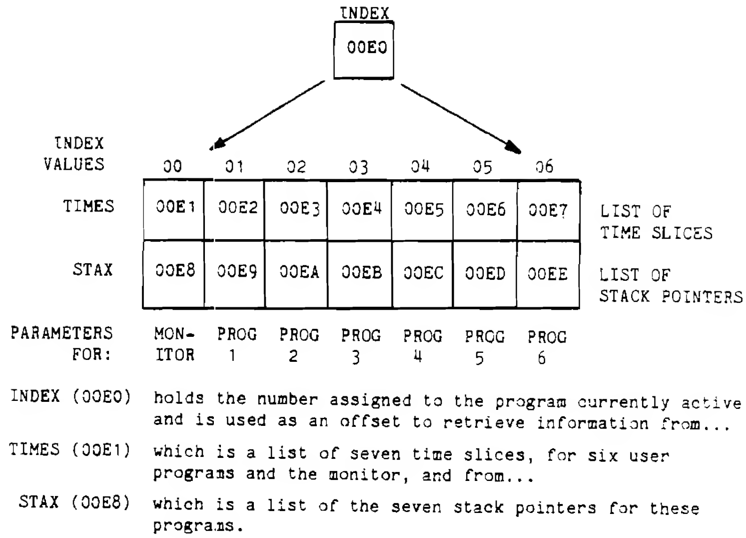


Figure 2: Use of page zero memory, 00E0-00EE

```
                ****************************************
                *  SAMPLE PROGRAMS FOR EXERCISING TSAR  *
                ****************************************
                *
                * 1ST SAMPLE PROGRAM MERELY COUNTS, IN HEX, THE NUMBER
                * OF TIMES IT IS ACTIVATED, STORING THE COUNT IN $0000,
                * WHICH SHOULD BE PRESET MANUALLY TO ZERO
                *
02E0                   ORG   $02E0
02E0 A9 00      START  LDAIM $00     CLEAR ACCUMULATOR AND USE IT TO
02E2 85 E1             STA   TIMES   ZERO OUT THE MONITOR TIME SLICE
02E4 A5 E1      LOOP   LDA   TIMES   FETCH THE MONITOR TIME SLICE, AND REMAIN IN
02E6 F0 FC             BEQ   LOOP    THIS LOOP AS LONG AS IT IS ZERO
02E8 E6 00             INC   COUNT   RECORD THE CHANGE, WHICH INDICATES
                                     COMPLETION OF ANOTHER
02EA 4C E0 02          JMP   START   CYCLE THROUGH THE QUEUE, AND REPEAT
                *
                * NOTE: USING THE MONITOR TIME SLICE AS A FLAG TO
                * INDICATE TO A PROGRAM WHEN IT HAS BEEN RECALLED
                * AND THEN REACTIVATED IS A USEFUL TRICK, BUT
                * AT A TIME
                *
                * 2ND SAMPLE PROGRAM ALSO COUNTS CYCLES, BUT IT
                * DOES SO IN BCD, KEEPING THE LEAST SIGNIFICANT
                * DIGITS IN 03FF, THE NEXT TWO IN 03FE,
                * AND SO ON.  IT CAN COUNT VERY HIGH.
                *
0200                   ORG   $0200
0200 F8         START2 SED           DECIMAL SPOKEN HERE
0201 38         STALL  SEC           SET CARRY SO ADDER WILL WORK PROPERLY
0202 A2 00             LDXIM $00     SET X REGISTER TO ZERO TO
0204 86 E1             STX   TIMES   ZERO THE MONITOR TIME SLICE
0206 A5 E1      LOOP2  LDA   TIMES   CHECK MONITOR TIME SLICE AND,
0208 F0 FC             BEQ   LOOP2   IF IT IS ZERO, KEEP ON CHECKIN'
020A CA         NEXT   DEX           TO FF, TO INDEX, INITIALLY, 03FF
020B BD 00 03          LDAX  X       GET CONTENTS OF 03XX
020E 69 00             ADCIM $00     ADD 1, SINCE THE CARRY IS SET
0210 9D 00 03          STAX  X       AND PUT IT BACK WHERE WE FOUND IT
0213 90 EC             BCC   STALL   WITHOUT CARRY, ADDITION IS FINISHED
                                     SO WAIT TILL NEXT TIME, OTHERWISE
0215 B0 F3             BCS   NEXT    BACK UP TO NEXT DIGIT & PROPAGATE CARRY

                * NOTE: THIS PROGRAM MAY BE USED WITH THE PREVIOUS
                * PROGRAM BY HAVING ONLY ONE OF THEM MESS WITH
                * TIMES AND HAVING THEM SHARE A RAM LOCATION AS
                * A FLAG.  ONE SETS THE FLAG, WHILE THE OTHER
                * RESETS THE FLAG AND LOOPS.
                *
```

While the monitor uses a single location, 00F2, for storing its only stack pointer, TSAR maintains a list, STAX, of stack pointers, one for each program on the queue. The six bytes of code from 178B to 1790 were included as an *after*-thought, *after* several foolish blunders on my part had let the system escape from my control. They merely guarantee the monitor's presence by forcing its time slice to "FF" if it is ever found at "00". Resorting to reset, to restore the monitor, is fun the first few times only. Nonetheless, these six bytes and the fifteen bytes used to service breakpoints may be deleted without otherwise affecting TSAR.

Bringing Up The System

Managing the TSAR system is quite a bit more complex than running a single program on the KIM, and several steps are required to put it into operation. The following sequence will generate a functioning TSAR system.

1. Verify that PB7 is connected to NMI (Figure 1).
2. Load TSAR into 1780-17BB, from keypad or tape.
3. If loading was from the keypad, verify correctness of code.
4. Set 17FA,B to point to 1780 and set 17FE,F to point to 17AC, providing the proper interrupt vectors for TSAR.
5. Load all locations from 00E0 through 00F1 with "00".
6. Press "RS", guaranteeing the stack pointer (monitor) at FF. If you are planning to use the DELAY subroutine from the ROM,

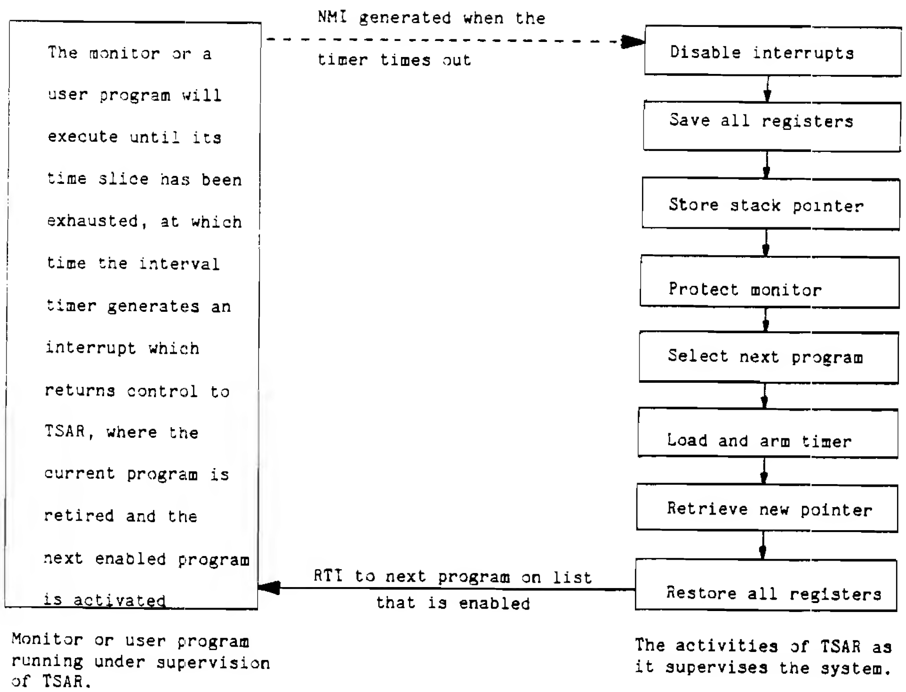


Figure 3: Flow of control in the TSAR system.

Table 1

Register Value Storage by the TSAR and by the KIM Monitor

TSAR Typical User Program Stack Locations	Register	KIM Monitor Saved	Dedicated KIM Page Zero RAM Locs.
01DA	Y		00F4
01DB	X		00F5
01DC	A		00F3
01DD	P		00F1
01DE	PCL		00EF (00FA)
01DF	PCH		00F0 (00FB)

Interrupt entry to TSAR (at 1780) or the Monitor (at 1C00) will store the MPU registers in the locations indicated above. Leaving TSAR via an RTI will restore these values to the MPU. Leaving the Monitor by using 'GO' restores the MPU from these RAM locations, except that the PCL and PCH are loaded from 00FA and 00FB (the pointer), respectively. To replace the original program counter into the MPU, the contents of 00EF and 00F0 are first transferred to 00FA and 00FB by pressing the 'PC' key, moving the Program Counter into the pointer.

*	Initial Y value	Initial X value	Initial A value	Initial flag values	Initial PCL (E0)	Initial PCH (02)
01D9	01DA	01DB	01DC	01DD	01DE	01DF

Figure 4: Initialization of a user program stack. The stack pointer initially points to 01D9, but since it is incremented before any values are pulled, the contents of 01D9 have no effect on the program.

remember that reset puts 17F3 to "FF". Also, if you are intending to use either port A or port B for output, you must reconfigure at this time, since reset configures all port lines as inputs.

7. Examine address 00E1, the monitor time slice. It will be "00".

8. Press "ST", NOT "GO"! we intentionally interrupt the monitor at this point to raise the activity to the TSAR system level. The value at 00E1 should now become "FF" as the monitor protection routine leaps in. If this does not happen, briefly address location 170C, another way to get an NMI pulse, and return to view 00E1. If it still does not read "FF", reset and check the startup sequence.

9. Now, assuming 00E1 is at "FF", try to key in "00". If the system rejects this, keeping "FF" instead, timesharing is in operation. If "00" is accepted in 00E1, generate another NMI (by examining 170C again) and verify timesharing as above.

10. As a final indication that timesharing is in operation, examine 00E8, the stack pointer for the monitor. Since the monitor is being interrupted, and not always at the same place, the value of 00E8 should change, probably flitting quickly from "F5" to "F7" and back. Any sign of flickering here verifies that TSAR is in charge and that timesharing is under way.

11. Key in a user program, noting that the monitor behaves as it always has. If you intend to load any user programs from tape, do so before step 8, as the timing changes under TSAR are not compatible with serial I/O. Assume that this first user program starts at location 02E0, as does the first of the sample programs, and that its stack extends downward from 01DF, leaving 32 bytes for the monitor, far more than it will ever need. This is program 1 (the monitor is program 0), so its initial stack pointer will go into STAX+1, 00E9. This stack will be accessed initially by lines 17A6 to 17AB of TSAR. Since TSAR first pulls register values for Y, X, and A, and then (with RTI) pulls three more values for P and PC, we must provide these six values immediately above the stack pointer. The values in the first four of these locations are used for the initial register contents when program 1 starts running.

As they are of no consequence for the sample programs, they may be set to "00" or left as found. However, the final two locations, 01DE and 01DF, hold the program counter for program 1 and must, in this case, be initialized to "E0" and "02" respectively, to provide the starting address, 02E0. Since the stack pointer must initially point to location 01D9, a "D9" is keyed into location 00E9.

12. Recheck the program code, stack values, and stack pointer.

13. Examine location 00E2 (TIMES +1), the time slice for program 1, and change it to a non-zero value. If your program does anything you can sense, you should be sensing it now. If it uses a counter, address the counter and watch it move. Return to 00E2 and vary the time slice, noting how the program execution speeds up and slows down. Change 00E1 and note its effect on the execution rate of your program. Enjoy it for a while, and then bring another program into the system. The procedure will be the same as above, from step 11 on, although a different location must be used for the stack, different initializing information must be placed on the stack, and the new stack pointer must be stored in a different position of STAX. Disable the first program; run either, both, or neither of them; play with it!

You are MASTER of your own timesharing system! The sample programs provided do not represent the full range of TSAR's potential. For one thing, keeping the monitor on-line prevents program-generated information from appearing on the display. With additional devices for output, the variety of interesting programs that can be run under TSAR is increased greatly.

For example, a memory-mapped video output can provide a very dramatic visual demonstration of timesharing. With a speaker connected as shown in the *Kim User Manual*, several programs may each toggle the speaker at rates determined by DELAY, a KIM monitor sub-routine at 1ED4 used for serial teleprinter I/O but also useful whenever a long software delay is required. They may also alter the DELAY parameters, (17F2,3), modify each other's time slices, and toggle the speaker port between input and output, producing a type of mayhem in the speaker that varies from WWII soundtracks, to tuba contests, to bouncing ball bearings, to almost-human-sounding arguments. Using "0F" to provide longer time intervals enhances this cacophony.

With several input devices (joysticks, keypads, even push buttons), TSAR permits the user connected to each device to have apparently sole use of the system, timesharing in the traditional, multi-user sense. With suitable ground rules established, the users could even play a version of "core war" in which each tries to get his (no doubt self-relocating) program to destroy the other programs before getting zapped by one of them. This has a vaguely evolutionary, survival-of-the-fittest undercurrent that keeps it from becoming too abstract.

Y	X	A	P	PCL	PCH		X	A	P	PCL	PCH
VALUE AT THE TIME OF BREAK	ITS OWN INDEX	(00)	FLAGS B, Z, ETC	EITHER B5 OR B7	(17)		THESE VALUES WERE STORED WHEN THE BREAK OCCURRED			THESE POINT TO THE INSTRUCTION AFTER BREAK OP-CODE	

Loaded when timer interrupts breakpoint routine. Loaded when break actually occurs.

Figure 5: Breakpoint stack contents. After recall, the pointer addresses the location below the Y value.

```
               *
               * 3RD SAMPLE PROGRAM ADVANCES THE POINTER (00FA,B)
               * TO DISPLAY SUCCESSIVE MEMORY LOCATIONS.  IT
               * INTERFERES SEVERELY WITH THE SIMULTANEOUS USE OF THE
               * MONITOR KEYPAD AND DISPLAY.
               *
0217 20 D4 1E  START3 JSR   DELAY  (PRESET 17F2 AND 17F3 TO
                            DETERMINE RATE OF DELAY
021A 20 63 1F         JSR   INCPT  INCREMENT THE POINTER, 00FA,B
021D A5 FA            LDA   POINTL GET THE LOW BYTE OF THE POINTER
                            AND USE IT AS THE VALUE
021F 85 E4            STA   TIMES  +03 VALUE OF THIS PROGRAM, SHOWN AS 3
0221 4C 17 02         JMP   START3 AND KEEP GOING
               *
               * NOTE: THIS PROGRAM BEHAVES UNPREDICTABLY.  IT MIGHT
               * DISABLE ITSELF, OR IT MIGHT SUSPEND THE ENTIRE SYSTEM
               * OR IT MIGHT KNOT.
               *

               ******************************
               * BREAKPOINT SERVICE ROUTINE *
               ******************************
               * (ENTRY FROM IRQ VECTOR)
               *
17AC                  ORG   $17AC
17AC 48        BREAK  PHA          SAVE A
17AD 8A               TXA          AND X
17AE 48               PHA          ON THE STACK
17AF A9 00            LDAIM $00    CLEAR A
17B1 A6 E0            LDX   INDEX  DETERMINE WHICH PROGRAM CAUSED BREAK
17B3 95 E1            STAX  TIMES  AND DISABLE IT
17B5 B5 E1     SLEEP  LDAX  TIMES  SLUMBER IN A LOOP UNTIL
17B7 F0 FC            BEQ   SLEEP  RECALLED BY TSAR
17B9 4C A8 17         JMP   GOBACK LET TSAR RESTORE X AND A BEFORE RETURN
               *
               * NOTE: THE RTI CODE AT $178B (IN TSAR) IS EXECUTED
               * TWICE AS A PROGRAM IS RE-AWAKENED AFTER A BREAK
               *
```

Keeping It Up

One problem the TSAR system does present is that, lacking proper safeguards, it is somewhat fragile. A single program, running amok, can bring down all of the others, including TSAR. Fortunately there are some methods for recovering gracefully from crashes, and even for averting many of them.

If the system seems to be misbehaving, it is a good idea to locate and disable the guilty program before it can interfere with other programs, the monitor, or TSAR. It is easy to disable any program simply by setting its time slice to zero. A record detailing what program is where on the queue and where the various stacks and stack pointers are located is very useful here. Once a program has been deactivated, it may be replaced on the queue with a different program, or it may be altered (repaired?) and then returned to service by a simple time slice change.

If disabling the suspect program fails to correct the system, the best procedure is to disable all user programs, and the faster the better. Then re-introduce them individually, testing them one at a time with the monitor. An externally generated IRQ signal is the quickest, cleanest way to disable all user programs, as it invokes the breakpoint service routine which disables the currently active program in an orderly manner. An interesting alternative is to have a special "shutdown" program ready but disabled. In case of trouble, enabling this program sends it into action to disable everything else and, finally, itself, in an orderly manner.

Triggering IRQ several times will null the time slices of all programs (monitor, however, remains, because it is not susceptible to IRQ), leaving each in a suspended state from which it can be returned to service by simply changing its time slice. This is a much less severe insult to the system than a reset produces, and it should be tried first, whenever dysfunction is suspected. Of course, if the system is hung in a loop with the I flag set, IRQ will be ignored and only a reset will affect the system.

If the system is 'hung', probably indicated by a stable, partially-lit display, the only option is a reset. Then, examine INDEX (00E0), to determine which program was running at the time of the reset interrupt. Disable that program and, if it was the whole problem, a "hot start" (set location 00E0 to "00" and then examine location 170C) should rekindle the system, minus the malfunctor. You can next locate the stack pointer for the disabled program and use it to determine the register contents (roughly) when it was last activated. Compare this with the response of the monitor at reset, which sets both the stack pointer and 00F2 to "FF", obliterating any traces of stack activity.

Breakpoints

Unique to TSAR, the provision for interrogating the code of a program while it is running can even be extended through the use of breakpoints, which themselves may be inserted into the program while it is running. This feature depends upon the coincidental good fortune that each 6502 branch instruction ends in a zero and can, therefore, be shifted left to the break code "00" without producing any dangerous intermediate code.

Recall that the timesharing procedure probably prevents entering, through the monitor, more than one hex character per time slice. For example, keying the break code over the code "4C" would first produce the interim code "C0" which would create havoc if executed before the second zero could be keyed in, during the next monitor time period. Changing a branch code, "X0", to a break code presents no such problems. Of course, there is the option of disabling the program, inserting the break, and reenabling it again; but inserting the break into "moving code" is more elegant and much more exciting.

When a break code is encountered, a non-maskable IRQ is generated, vectoring control to the BSR code, presented in the listing. This routine first saves the A and X registers on the stack used by the interrupted program. It then sets the time slice of the interrupted program to zero, and loops on this condition until the current time slice expires and the program is recalled by TSAR. The user can detect the occurrence of a break by watching the location holding its time slice, or he can provide a watchdog program to monitor this value and produce a signal when it detects a zero. TSAR will bypass this program on subsequent cycles through the queue, because of its null time slice, so the idle program, its breakpoints, and its stack may be examined and altered at leisure, until it is ready to be run again.

At that time, merely keying in a non-zero time slice for the program signals to TSAR that it is to be reactivated, when its turn comes. Although reactivation returns it to the loop where it was sleeping before recall, the loop condition (time slice = zero) has been changed, so the program can escape from this loop and reenter its old code at the next instruction after the break.

Since the procedure for bringing a program back from a break is somewhat involved, requiring as it does the unnesting of two different interrupts, a closer look might be worthwile. First TSAR, at line 179D, discovers that the program is again enabled, so its time slice is loaded into the interval timer. Then the stack pointer for this program is brought in from STAX. It will point to the location just below that where the Y register was stored. Registers Y, X, and A are loaded from this stack, and RTI restores the flag register, in which the Z flag is SET, and returns control at line 17B5 or 17B7 of the breakpoint routine. This is the stalling loop where the program idled from the break interrupt until its former time slice expired and it was recalled by TSAR.

Now, however, its time slice has been adjusted from zero, and when this is discovered the loop is abandoned and control goes once more to TSAR's exit routine, this time at 17A8. The X and A values from before the break are brought in, and the second pass through RTI restores P and PC, returning control to the user program at the instruction following the branch/break code. This entire procedure is carried out with no effect on the other programs operating under TSAR, except that each runs a bit more slowly when this program returns to service and again requires a slice of the MPU's time.

Caveat Computor

Because of significant differences between operating under TSAR and operating under the KIM monitor, a few warnings are in order. Although most have been mentioned before, they are collected here for emphasis and elaboration. Programs running simultaneously under TSAR, including the monitor, must not normally share RAM storage or use common subroutines unless they are fully reentrant. This restricts user programs from calling the keypad and display routines if the monitor is enabled, and monitor RAM locations, like the pointer at 00FA,B must be scrupulously avoided. However, it is possible to bring up the TSAR system without an enabled monitor, permitting user programs to use the monitor utility routines. Simply altering the monitor protect code and then disabling the monitor is an inelegant but easy way to manage this. It does, however, fill one place on the queue with a dead monitor.

A better procedure is to set up all the stacks, time slices, and pointers in advance, initiate the execution of a single user program from the monitor (with "GO"), and then use "ST" to leap up to TSAR. Although this approach sacrifices interactive control of the system, that may be prevented by giving up the breakpoint routine and re-directing IRQ to the monitor at 1C00. An external device (switch?) that can deliver an IRQ might now restore the monitor on-line. Note that this procedure differs from a recall by TSAR, in that the registers of the interrupted program are saved in monitor RAM instead of the program stack, meaning that the monitor has, for the time being, replaced one of the user programs on the queue. When the monitor is no longer needed, "PC" followed by "GO" will switch them back again, putting the monitor out of and the user program back into circulation.

A disadvantage of this procedure is that, without additional control hardware, the program which is replaced by the monitor will be selected by chance, and several attempts may be needed to locate a suitable candidate, one you are willing to have idle as long as the monitor is in use. To minimize repeated blind interrupts and restarts of the system, disable all of the programs that you wish to keep running the first time you IRQ the system into the monitor. This greatly increases the chance that, on the

next interrupt, a non-essential program will be replaced by the monitor, and then the disabled programs can be reenabled. I prefer, instead, to retain continuous monitor presence and have my user programs do their I/O through ports rather than through the keypad and display routines.

Because of the changes in timing introduced, serial I/O drivers, such as the cassette and serial teleprinter routines in the ROM, cannot be expected to operate properly under TSAR.

For more than six user programs, references to STAX, TIMES, and INDEX will need to be changed in TSAR, to reflect the re-organization of page zero memory use.

One of the most bizarre malfunctions that can occur under TSAR is to have more than one copy of the monitor concurrently active. Since the code is not reentrant, the multiple copies share RAM locations and interact oddly, producing such symptoms as:

a. Keystroke double entry. This may be nice for bbookkeepers, but it makes it very difficult to address location 0327 when pressing the "3" key inserts two nibbles of "3", while the "+" key advances two cells at a time.

b. Total or sporadic failure to respond to certain keystroke commands, as one copy of the monitor receives the command; but, before it can finish executing it, the other copy garbles up the message.

An intriguing challenge, at this point, is to locate and disable the imposter—the marauding mock monitor—before it brings down the system altogether. My record of two successes in five trys is more impressive than it sounds.

The possibility of numerous heirarchy violations exists under TSAR because, in the absence of protectable memory regions (ROM doesn't count here), any executing program is considered the equal of any other. This permits a lowly user program, intentionally or otherwise, to plunder page twenty-three and wound or altogether destroy beloved TSAR. He may even manage to wrest control of the system, gaining thereby a sort of immortality, by preventing the changing of INDEX or by disabling all competition. The opportunities for such exotic malfunctioning are vast, but they are easy to avoid and the interest they contribute far outweighs whatever minor annoyance they might occasionally produce. In fact, they can be a source of very interesting diagnostic opportunities. For instance, imagine trying to reestablish control in a situation where monitor monitors monitor.

RTI

As I mentioned earlier, neither the design nor the operation of TSAR is overly complicated. In spite of the enormous increase in capability that TSAR brings to KIM, the system is really quite simple to bring up and to operate. In fact, except for some flickering of the display, the monitor behaves as if it were in charge, rather than operating under the supervision of TSAR. Moreover, I have found that any apparent malfunctioning of TSAR could eventually be traced to carelessness on my part—in running a flawed program or in failing to initialize a pointer-stack combination properly.

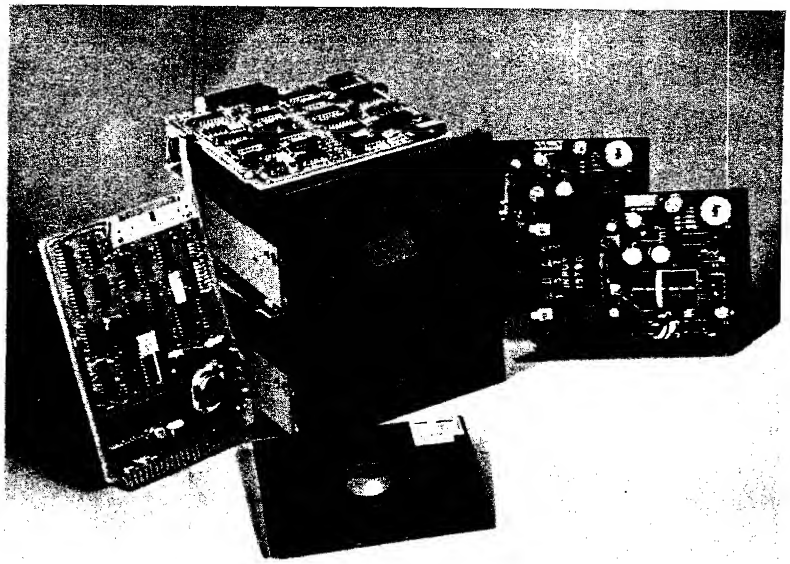
I assume that this system is easily adapted for use on other 6502 computers, and I would like to hear from anyone who brings it up on an AIM, SYM, or other 6502 machine, or who finds interesting, useful, or entertaining applications for it. How about a memory mapped display routine providing current information regarding the system status, like: number of currently enabled programs, disabled (i.e. available for use) INDEX values, percentage of running time alloted to each enabled program, maximum stack depth attained by each program (could head off disasters). Of course, this program would be on the queue and would be reporting on itself as well as on the others. With all that vacant ROM space from 1A96 to 1BF9, I wish I knew a way to hide TSAR up there, out of danger from peasant programs and proletarian programmers, but ready to take command of a timesharing system when summoned.

HDE inc.

BOX 120
ALLAMUCHY, N.J. 07820
201-362-6574

HUDSON DIGITAL ELECTRONICS INC.

THE HDE MINI-DISK SYSTEM



VERSIONS

KIM
TIM
AIM 65 – 4th Qtr. '79
SYM – 1st Qtr. '80

SINGLE DRIVE	**$ 795.00**
DUAL DRIVE	**$1195.00**

Complete with all hardware. Interconnecting cables, FODS, text editor and user and installation manuals.

The HDE DM816-MD1 Mini Disk System is the peripheral you have been waiting for. No longer bounded by long and unreliable cassette saves and loads, your computer becomes a sophisticated system for program development or general purpose use. With the HDE Mini-Disk you load and save programs in seconds, not minutes or hours. And, since all transfers to and from the Mini-Disk are verified for accuracy, the data will be there when you need it.

The HDE DM816-MD1 Mini-Disk has been "systems" engineered to provide a complete and integrated capability. Software and hardware have been built as a team using the most reliable components available. The systems software includes the acclaimed and proven HDE File Oriented Disk System and Text EDitor, requiring only 8K for the operating software and overlay area. Systems expanding programs available include the two-pass HDE assembler, the Text Output Processing System and Dynamic Debugging Tool. Hardware includes a Western Digital 1771 based controller in a state-of-the-art 4½ x 6½" card size, Shugart SA 400 drive and the Alpha power supply.

The storage media for the DM816-MD1 is the standard, soft sectored 5¼" mini diskette readily available at most computer stores, and HDE has designed the system so that the diskettes rotate only during disk transactions, favorably extending media life. A disk formatter routine included with the system, formats the diskettes, verifies media integrity by a comprehensive R/W test and checks drive RPM. Additional utilities provide ascending or descending alpha numeric sort, disk packing, text output formatting, file renaming, file addressing and other capabilities.

HDE PRODUCTS. BUILT TO BE USED WITH CONFIDENCE.

AVAILABLE DIRECT OR FROM THESE FINE DEALERS:

JOHNSON COMPUTER	PLAINSMAN MICROSYSTEMS	ARESCO	LONG ISLAND COMPUTER GENERAL STORE	LONE STAR ELECTRONICS
Box 523	Box 1712	P.O. Box 43	103 Atlantic Avenue	Box 488
Medina, Ohio 44256	Auburn, Ala. 36830	Audubon, Pa. 19407	Lynbrook, N.Y. 11563	Manchaca, Texas 78652
216-725-4560	800-633-8724	215-631-9052	516-887-1500	612-282-3570

Interfacing the CI-812 to the KIM

If you want to add I/O capabilities to your KIM, then consider the CI-812 I/O board and its abilities.

Jim Dennis
2305 Pinecrest
Nacogdoches, TX 75962

The Percom CI-812 I/O board contains a full-duplex data terminal interface (RS-232) and a cassette interface that can load and dump data at rates up to 2400 baud. The CI-812 comes with 8080 software and is useless to 6502 owners, as is. I have interfaced a CI-812 to a KIMSI 6502 to S-100 motherboard, and I have written software that loads and dumps to the CI-812 from a KIM.

There are several reasons why I wanted to add this board to my I/O library. First, under the right conditions, data transfer can take place very quickly compared to the standard 10 cps of the KIM, and the rates are easily controlled in the software.

Second, if the user is interested in building a terminal to communicate with a big computer or with another small computer via a modem, all that is needed additionally is a modem ($125 for a Pennywhistle), a TVT-6 ($35), and a video monitor ($150), or a converted black and white TV, to turn the KIM into a full-fledged intelligent terminal.

Third, data received from magnetic tape is self-clocked with a signal extracted from the data. Speed variations in the tape drive and baud rate changes are thereby eliminated as sources of error.

The KIMSI generates S-100 bus signals and the decode enable from the signals on the expansion connector of the KIM. S-100 signals of the bus master type used by the CI-812, in addition to the tri-stated address lines and the DO and DI lines, include PWR, PDBIN, SINP, and SOUT. PWR is an active low signal denoting stable data on the DO lines, PDBIN is an active high signal that enables the DI lines, while SINP and SOUT are active high signals that indicate the addressing of an I/O device.

The CI-812 does not directly interface with the KIMSI because of timing problems associated with SINP, SOUT, and the DO buffer enable. Also, the 2 MHz clock pulse required by the CI-812 is not generated by the KIMSI. The procedure for overcoming these problems is:

1. Jumper the 1 MHz enable from finger 62 of the KIMSI to finger 49 of the KIMSI.
2. Bypass the first divide-by-two stage of the clock by bending up pin 5 IC9 of the CI-812 and jumpering pin 5 IC12 to pin 8 IC3.
3. Create a new signal, which I call SNOUT, that goes high whenever an I/O device is addressed. SNOUT is available at pin 10 IC9 of the KIMSI. Jumper it to finger 96 of the KIMSI.
4. Bypass the OR-INVERT of SINP and SOUT by the CI-812 by jumpering finger 96 of the KIMSI to pin IC20 of the CI-812 and by bending up pin 4 IC2 of the CI-812.
5. Permanently enable the DO buffers by jumpering pin 12 IC14 of the CI-812.

This completes the hardware revision of the KIMSI and the CI-812. The CI-812 outputs bi-phase (Manchester) code consisting of bursts of 2400 Hz square waves for a logic one and 1200 Hz square waves for a logic zero. Impressing unfiltered square waves on magnetic tape and then reading them involves a double differentiation process that can cause errors at high baud rates. For this reason, computer grade tape and baud rates of not more than 300 are recommended. Three hundred baud is known as the Kansas City standard.

The program shown is a checksum loader-dump routine which I have written for the KIM-PERCOM-KIMSI combination.The KIMSI uses memory mapping to address I/O devices, reserving address range FOXX for I/O devices.

My PERCOM board is addressed at F0EX; X = 1, 2. The program follows the format of the KIM cassette loader and dump, loading block headers and EOT's in Hex and all else in ASCII. No SYN characters are necessary. When a program has been dumped, the monitor takes over at address 0000. If a program has loaded correctly, the address display will light at 0000 also. If an illegal hex character has been encountered, meaning that the tape has been read incorrectly, the display will light at the starting address of the loader.

For example, if two ASCII characters are decoded to a J6, which is supposed to be a hex byte, then the tape has been read incorrectly. If a checksum error occurs, then the display will light with the calculated checksum high or low byte repeated twice in the address display.

The ASCII — hex checksum load and dump routine for the KIM uses the following KIM monitor subroutines: VEB, INTVEB, CHKT, INCVEB and PACKT. The CI-812 is addressed at F0E1 and F0E2 in the program listing. The ASCII — hex dump starts at 0000 and loads at 0070. To change to another location, modify locations denoted by "*" to reference the new page.

SUBROUTINE	FUNCTION
OUTCHR	DUMPS ASCII CHR ON TAPE
OUTBYTC	DUMPS 1HEX BYTE AS 2 ASCII CHR AND INCREMENTS CHKSUM
OUTBYT	SAME AS ABOVE BUT DOES NOT INC CHKSUM
LDASCII	LOADS 1 ASCII CHR
HEXTOAS	CONVERTS 1/2 HEX BYTE TO ASCII

```
0000 A9 AD              LDAIM AD      INITIALIZE VEB AS DUMP
0002 8D EC 17           STA VEB
0005 20 32 19           JSR INTVEB    INITIALIZE VEB
0008 A9 2A              LDAIM ASCII'*'  ASCII SYNC
000A 20 56 00*          JSR OUTCHR    OUTPUT BLOCK HEADER
000D AD F9 17           LDA ID
0010 20 03 01*          JSR OUTBYT    OUTPUT ID WITHOUT CHKS
0013 AD F5 17           LDA SAL       STARTING ADDRESS LOW
0016 20 00 01*          JSR OUTBYTC   OUTPUT WITH CHKSUM
0019 AD F6 17           LDA SAH       STARTING ADDRESS HIGH
001C 20 00 01*          JSR OUTBYTC
001F AD ED 17  STRT     LDA VEB + 1   GET CURRENT ADD. LOW
0022 CD F7 17           CMP EAL       CMP WITH ENDING ADD. L
0025 AD EE 17           LDA VEB + 2   GET CURRENT ADD. HIGH
0028 ED F8 17           SBC EAH       SBC ENDING ADD. HIGH
002B 90 1A              BCC DUMP2     DO THEY AGREE?
002D A9 2F              LDA ASCII'/'  YES, LDA ASCII SLASH
002F 20 56 00*          JSR OUTCHR    OUTPUT EOT
0032 AD E7 17           LDA CHKSUML   GET CHKSUM LOW
0035 20 03 01*          JSR OUTBYT    OUTPUT
0038 AD E8 17           LDA CHKSUMH   GET CHKSUM HIGH
003B 20 03 01*          JSR OUTBYT    OUTPUT
003E A9 00              LDA 00
0040 85 FA              STAZ POINTL
0042 85 FB              STAZ POINTH
0044 4C 4F 1C           JMP START     ALL OK, RETURN TO MON
0047 20 EC 17  DUMP2    JSR VEB       PICK UP NEXT BYTE
004A 20 00 01*          JSR OUTBYTC   OUTPUT
004D 20 EA 19           JSR INCVEB    INC CURRENT ADDRESS
0050 4C 1F 00*          4C STRT
0053 EA                 NOP
0054 EA                 NOP
0055 EA                 NOP
0056 48                 PHA           SAVE BYTE
0057 A9 03              LDAIM 03      LDA SELECT CODE
0059 8D E0 F0           STA CAS-SEL   SELECT CASSETTE MODE
005C AD E1 F0           LDA UARTOUT   READ UART TO CLEAR
005F AD E0 F0  CLEAR    LDA STATUS    READ STATUS
0062 29 80              ANDIM 80      MASK STATUS BIT
0064 F0 F9              BEQ CLEAR     LOOP IF STILL TRANSMIT
0066 68                 PHA           RESTORE BYTE
0067 8D E1 F0           STA CASOUT    OUTPUT TO UART
006A 60                 RTS
006B EA                 NOP
006C EA                 NOP
006D EA                 NOP
006E EA                 NOP
006F EA                 NOP           LOADER BEGINS HERE
0070 A9 8D              LDAIM 8D      SET UP VEB AS LOADER
0072 8D EC 17           STA VEB
0075 20 32 19           JSR INTVEB    INITIALIZE VEB AS LOADER
0078 A9 4C              LDAIM 4C
007A 8D EF 17           STA VEB +3
007D A9 C6     *        LDAIM C6
007F 8D F0 17           STA VEB + 4
0082 A9 00     *        LDAIM 00      LOADER RETURNS FROM VEB
0084 8D F1 17           STA VEB + 5   WITH JMP TO LOC. 00C6
0087 20 32 01* RDY?     JSR LDASCII   LOOK FOR BLOCK HEADER
008A C9 2A              CMPIM '*'     IS IT A SYNC?
008C F0 02              BEQ GETBYT    YES, PICK UP NEXT 2 CHARAC.
008E D0 F7              BNE RDY?      NO, LOOK AGAIN
0090 20 23 01* GETBYT   JSR ASCIIHEX  GET NEXT 2 CHAR. AND CONVERT
0093 CD F9 17           CMP ID        IS IT THE RIGHT BLOCK?
0096 F0 02              BEQ GO        YES, GET FIRST CHARACTER
0098 D0 F6              BNE GETBYT    NO, KEEP LOOKING FOR ID
009A 29 23 01* GO       JSR ASCIIHEX  GET BYTE AND CONVERT TO HEX
009D 20 4C 19           JSR CHKSUM    INC CHECKSUM
00A0 8D ED 17           STA VEB + 1   STORE        CHKSUM LOW
00A3 20 23 01*          JSR ASCIIHEX
00A6 20 4C 19           JSR CHKSUM
00A9 8D EE 17           STA VEB + 2   STORE CHKSUM HIGH
00AC A2 02     GO2      LDXIM 02      LDX CHAR. COUNTER
00AE 20 32 01* GO1      JSR LDASCII   GET ASCII CHAR.
00B1 C9 2F              CMPIM EOT     IS IT EOT?
00B3 F0 17              BEQ CONT      YES, FINISH
00B5 20 00 1A           JSR PACKT     NO, PACK ASCII AS HEX
00B8 F0 03              BEQ VALASC    BEQ VALID ASCII CHAR.
00BA 4C 4F 1C           JMP START     ERROR EXIT
00BD CA        VALASC   DEX           DEC. CHAR. COUNTER
00BE D0 EE              BNE GO1       GET 2ND CHAR.
00C0 20 4C 19           JSR CHKSUM    INC CHKSUM
00C3 4C EC 17           JMP VEB       MOVE SA TO VEB
00C6 20 EA 19           JSR INCVEB    INC. CURRENT ADD.
00C9 4C AC 00*          JMP GO2       LOOP BAK FOR MORE CHAR.
00CC 20 23 01* CONT     JSR ASCIIHEX  GET CHKSUM
00CF F0 03              BEQ CKOK?     IT IT VALID HEX?
00D1 4C 2B 19           JMP ERROREX   NO, EXIT
00D4 CD E7 17  CKOK?    CMP CHKSUML   YES, COMPARE WITH CALC. CHKS
00D7 F0 03              BEQ OK        CHKSUM LOW AGREES
00D9 4C 2B 19  BADNEWS  JMP ERROREX   CHKSUM LOW DOES NOT AGREE
00DC 20 23 01* OK       JSR ASCIIHEX  GET CHKSUM HIGH
00DF CD E8 17           CMP CHKSUMH   COMPARE WITH CALC. CHKSUMH
00E2 D0 F5              BNE BADNEWS   CHKSUM HIGH DOES NOT AGREE
00E4 A9 00              LDAIM 00
00E6 4C 2B 19           JMP NORMEX    CHKSUM AGREES
0100 20 4C 19           JSR CHKSUM    CALC. CHKSUM
0103 A8                 TAY           SAVE BYTE
0104 4A                 LSRA
0105 4A                 LSRA
0106 4A                 LSRA
0107 4A                 LSRA          SHIFT OUT LSB
0108 20 13 01 *         JSR HEXTOAS   CONVERT TO ASCII
010B 98                 TYA           RESTORE BYTE
010C 20 13 01 *         JSR HEXTOAS   OUTPUT MSB AS ASCII
010F 98                 TYA           RESTORE BYTE
0110 60                 RTS
0111 EA                 NOP
0112 EA                 NOP
0113 29 0F              ANDIM 0F      MASK MSB
0115 C9 0A              CMPIM 0A
0117 18                 CLC
0118 30 02              BPL CONV
011A 69 07              ADCIM 07
011C 69 30     CONV     ADCIM 30      CONVERT TO ASCII
011E 20 56 00*          JSR OUTCHR    OUTPUT AS ASCII
0121 60                 RTS
0122 EA                 NOP
0123 20 32 01*          JSR LDASCII   READS UART
0126 20 00 1A           JSR PACKT     PACKS 2 ASCII CHAR.
0129 20 32 01*          JSR LDASCII   AS 1 HEX BYTE
012C 20 00 1A           JSR PACKT
012F 60                 RTS
0130 EA                 NOP
0131 EA                 NOP
0132 A9 01              LDAIM 01      CODE FOR CASSETTE LOAD
0134 8D E0 F0           STA UART      OUTPUT TO UART
0137 AD E1 F0  LOOP     LDA UARTOUT   CLEAR UART
013A AD E0 F0           LDA FLAG      READ STATUS
013D 29 40              ANDIM 40      MASK STATUS BIT
013F F0 F9              BEQ LOOP      IF NOT READY, WAIT
0141 AD E1 FC           LDA DATA      LOAD ASCII CHAR.
0144 60                 RTS           RETURN
```

Microbes

AMPERSORT

Alan G. Hill
12092 Deerhorn Drive
Cincinnati, OH 45240

I apologize to MICRO readers for the errors in the listing of AMPERSORT published in MICRO 14:39. The problem was a result of including the first five pages of an earlier version with the last two pages of a later version to which lines 3940 thru 3946 were added. This caused, as many readers discovered, the object address of some of the preceding code to be incorrect. Attached is a listing of the correct object code. Anyone wishing to receive an improved version on cassette may do so by sending $5.00 to me at the above address.

Several people have asked if AMPERSORT can be used with Applesoft in RAM rather than ROM. With the following changes it can:

Routine	ROM Addr.	RAM Addr.
FRMNUM	$DD67	$156A
GETADR	$E752	$1F49
GETBYT	$E6F8	$1EEF
SNER	$DEC9	$16CC

The Applesoft RAM BASIC program must also include the following statements that must be executed prior to the first '&SRT' command:

POKE 2142,244: POKE 2143,3

The specific changes to AMPERSORT for Applesoft RAM are:

Address	ROM Ver.	RAM Ver.
$5269	67	6A
$526A	DD	15
$526C	52	49
$526D	E7	1F
$527A	67	6A
$527B	DD	15
$527D	52	49
$527E	E7	1F
$52A9	C9	CC
$52AA	DE	16
$52B4	F8	EF
$52B5	E6	1E
$52C0	F8	EF
$52C1	E6	1E

```
*5200.5589

5200- 48 20 E6 54 68 A2 00 DD
5208- 2C 55 D0 46 20 B1 00 E8
5210- E0 05 D0 F3 A2 00 F0 03
5218- 20 B1 00 C9 2C F0 0A 9D
5220- 72 55 E8 E0 10 D0 F1 F0
5228- 29 CA BD 72 55 C9 24 F0
5230- 24 C9 25 D0 15 A2 01 A9
5238- 80 1D 72 55 9D 72 55 CA
5240- 10 F5 A9 02 85 EC A9 01
5248- D0 19 A9 05 85 EC A9 02
5250- D0 11 4C A5 52 A9 80 0D
5258- 73 55 8D 73 55 A9 03 85
5260- EC A9 00 85 F1 20 B1 00
5268- 20 67 DD 20 52 E7 A5 50
5270- 85 DE A5 51 85 DF 20 B1
5278- 00 20 67 DD 20 52 E7 A5
5280- 50 85 D4 18 69 01 85 E0
5288- A5 51 85 D5 69 00 85 E1
5290- A5 F1 D0 59 F0 15 A2 00
5298- BD 31 55 09 80 20 ED FD
52A0- E8 E0 17 D0 F3 20 09 55
52A8- 4C C9 DE A0 00 8C 89 55
52B0- 20 B1 00 20 F8 E6 CA AC
52B8- 89 55 96 E2 20 B1 00 20
52C0- F8 E6 AC 89 55 96 E7 20
52C8- B1 00 90 D9 C9 44 F0 04
52D0- A9 FF 30 02 A9 00 99 82
52D8- 55 C8 8C 89 55 20 B1 00
52E0- C9 29 F0 06 C9 2C F0 C8
52E8- D0 BB 8C 88 55 20 B1 00
52F0- D0 B3 A0 00 B1 6B CD 72
52F8- 55 D0 08 C8 B1 6B CD 73
5300- 55 F0 2B 18 A0 02 B1 6B
5308- 65 6B 48 C8 B1 6B 65 6C
5310- 85 6C 68 85 6B C5 6D A5
5318- 6C E5 6E B0 03 4C F2 52
5320- A2 02 BD 72 55 9D 3B 55
5328- CA 10 F7 4C 96 52 18 A5
5330- 6B 69 07 85 52 A5 6C 69
5338- 00 85 53 A5 DE 85 50 A5
5340- DF 85 51 A5 EC 85 54 A9
5348- 00 85 55 20 63 FB A5 50
5350- 85 D6 A5 51 85 D7 4C 66
5358- 53 18 A5 D6 65 EC 85 D6
5360- A5 D7 69 00 85 D7 A0 01
5368- B1 D6 85 D8 C8 B1 D6 85
5370- D9 18 A5 D6 65 EC 85 DA
5378- A5 D7 69 00 85 DB 18 A5
5380- DE 69 01 85 ED A5 DF 69
5388- 00 85 EE 4C 9B 53 18 A5
5390- DA 65 EC 85 DA A5 DB 69
5398- 00 85 DB A0 01 B1 DA 85
53A0- DC C8 B1 DA 85 DD A5 F1
53A8- F0 03 4C 2F 54 A0 00 B1
53B0- D6 F0 52 85 EF B1 DA F0
53B8- 4C 85 F0 A2 00 B4 E2 BD
53C0- 82 55 30 0C B1 D8 D1 DC
53C8- B0 14 20 C1 54 4C 05 54
53D0- B1 D8 D1 DC 90 2F F0 19
53D8- 20 C1 54 4C 05 54 D0 25
53E0- C8 C4 EF F0 06 C4 F0 F0
53E8- 16 90 0F C4 F0 90 E9 F0
53F0- 0E C8 C4 EF F0 09 C4 F0
53F8- F0 DE 98 D5 E7 D0 C0 E8
5400- EC 88 55 D0 B8 E6 ED D0
5408- 02 E6 EE A5 ED C5 E0 A5
5410- EE E5 E1 90 14 E6 DE D0
5418- 02 E6 DF A5 DE C5 D4 A5
5420- DF E5 D5 90 07 20 09 55
5428- 60 4C 8E 53 4C 59 53 18
5430- 6A B0 03 4C 6D 54 A0 01
5438- B1 D6 D1 DA 88 B1 D6 F1
5440- DA 90 22 B1 D6 51 DA 30
5448- BC C8 B1 DA 48 88 B1 DA
5450- 48 B1 D6 91 DA C8 B1 D6
5458- 91 DA 88 68 91 D6 C8 68
5460- 91 D6 4C 05 54 B1 D6 51
5468- DA 30 DE 10 98 A0 00 B1
5470- D6 D1 DA 90 0B F0 02 B0
5478- 1D C8 C0 05 D0 F1 F0 3E
5480- A0 01 B1 D6 31 DA 11 DA
5488- 30 20 88 B1 DA D0 2F C8
5490- B1 D6 10 16 30 28 A0 01
5498- B1 D6 31 DA 11 D6 30 1E
54A0- 88 B1 D6 D0 05 C8 B1 DA
54A8- 10 14 A0 04 B1 D6 48 88
54B0- 10 FA C8 B1 DA 91 D6 68
54B8- 91 DA C0 04 D0 F4 4C 05
54C0- 54 A0 00 B1 D6 48 C8 A5
54C8- D8 91 DA C8 A5 D9 91 DA
54D0- A5 DD 91 D6 85 D9 88 A5
54D8- DC 91 D6 85 D8 88 B1 DA
54E0- 91 D6 68 91 DA 60 A2 00
54E8- B5 D0 9D 48 55 E8 E0 22
54F0- D0 F6 A5 6B 8D 70 55 A5
54F8- 6C 8D 71 55 A2 00 B5 50
5500- 9D 6A 55 E8 E0 06 D0 F6
5508- 60 A2 00 BD 48 55 95 D0
5510- E8 E0 22 D0 F6 AD 70 55
5518- 85 6B AD 71 55 85 6C A2
5520- 00 BD 6A 55 95 50 E8 E0
5528- 06 D0 F6 60 53 52 54 23
5530- 28 8D 56 41 52 49 41 42
5538- 4C 45 20 20 20 20 20 4E
5540- 4F 54 20 46 4F 55 4E 44
5548- 00 00 00 00 00 00 00 00
5550- 00 00 00 00 00 00 00 00
5558- 00 00 00 00 00 00 00 00
5560- 00 00 00 00 00 00 00 00
5568- 00 00 00 00 00 00 00 00
5570- 00 00 00 00 00 00 00 00
5578- 00 00 00 00 00 00 00 00
5580- 00 00 00 00 00 00 00 00
5588- 00 00
```

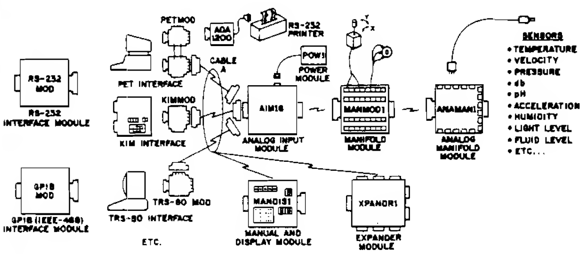
Note on Charles Husband's Speech Processor for the PET MICRO 16:41

Readers interested in obtaining additional information about the **Data—Boy Speech Processor** should contact Jim Anderson at:

MIMIC Electronics
Box 921
Acton, MA 01720

Data Acquisition Modules

DAM SYSTEMS

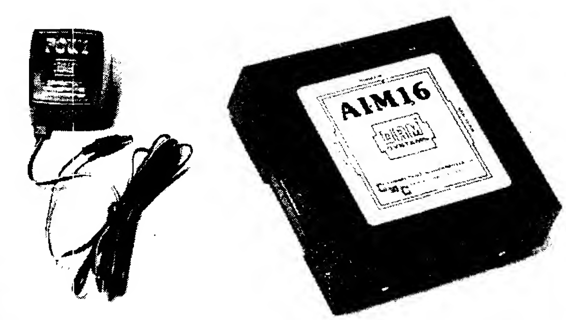


The world we live in is full of variables we want to measure. These include weight, temperature, pressure, humidity, speed and fluid level. These variables are continuous and their values may be represented by a voltage. This voltage is the analog of the physical variable. A device which converts a physical, mechanical or chemical quantity to a voltage is called a sensor.

Computers do not understand voltages: They understand bits. Bits are digital signals. A device which converts voltages to bits is an analog-to-digital converter. Our AIM16 (Analog Input Module) is a 16 input analog-to-digital converter.

The goal of Connecticut microComputer in designing the DAM SYSTEMS is to produce easy to use, low cost data acquisition modules for small computers. As the line grows we will add control modules to the system. These acquisition and control modules will include digital input sensing (e.g. switches), analog input sensing (e.g. temperature, humidity), digital output control (e.g. lamps, motors, alarms), and analog output control (e.g. X-Y plotters, or oscilloscopes).

Analog Input Module



The AIM16 is a 16 channel analog to digital converter designed to work with most microcomputers. The AIM16 is connected to the host computer through the computer's 8 bit input port and 8 bit output port, or through one of the DAM SYSTEMS special interfaces.

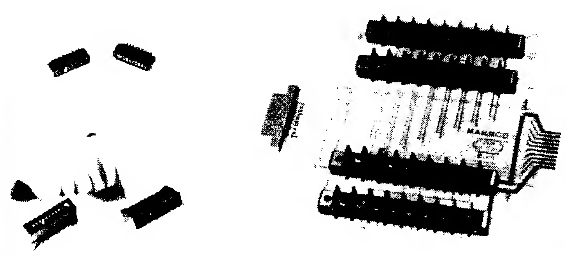
The input voltage range is 0 to 5.12 volts. The input voltage is converted to a count between 0 and 255 (00 and FF hex). Resolution is 20 millivolts per count. Accuracy is 0.5% ± 1 bit. Conversion time is less than 100 microseconds per channel. All 16 channels can be scanned in less than 1.5 milliseconds.

Power requirements are 12 volts DC at 60 ma.

The POW1 is the power module for the AIM16. One POW1 supplies enough power for one AIM16, one MANMOD1, sixteen sensors, one XPANDR1 and one computer interface. The POW1 comes in an American version (POW1a) for 110 VAC and in a European version (POW1e) for 230 VAC.

AIM16 . . . $179.00
POW1a . . . $ 14.95
POW1e . . . $ 24.95

Connectors



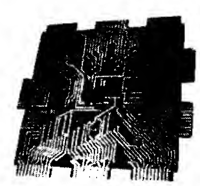
The AIM16 requires connections to its input port (analog inputs) and its output port (computer interface). The ICON (Input CONnector) is a 20 pin, solder eyelet, edge connector for connecting inputs to each of the AIM16's 16 channels. The OCON (Output CONnector) is a 20 pin, solder eyelet edge connector for connecting the computer's input and output ports to the AIM16.

The MANMOD1 (MANifold MODule) replaces the ICON. It has screw terminals and barrier strips for all 16 inputs for connecting pots, joysticks, voltage sources, etc.

CABLE A24 (24 inch interconnect cable has an interface connector on one end and an OCON equivalent on the other. This cable provides connections between the DAM SYSTEMS computer interfaces and the AIM16 or XPANDR1 and between the XPANDR1 and up to eight AIM16s.

ICON . . . $ 9.95
OCON . . . $ 9.95
MANMOD1 . . . $59.95
CABLE A24 . . . $19.95

XPANDR1



The XPANDR1 allows up to eight AIM16 modules to be connected to a computer at one time. The XPANDR1 is connected to the computer in place of the AIM16. Up to eight AIM16 modules are then connected to each of the eight ports provided using a CABLE A24 for each module. Power for the XPANDR1 is derived from the AIM16 connected to the first port.

XPANDR1 . . . $59.95

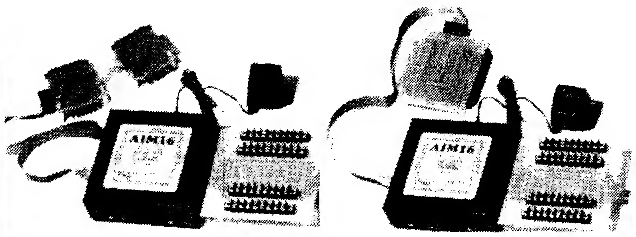
TEMPSENS

This module provides two temperature probes for use by the AIM16. This module should be used with the MANMOD1 for ease of hookup. The MANMOD1 will support up to 16 probes (eight TEMPSENS modules).

Resolution for each probe is 1°F.

TEMPSENS2P1 (-10°F to 120°F) . . . $49.95

Computer Interfaces and Sets



For your convenience the AIM16 comes as part of a number of sets. The minimum configuration for a usable system is the AIM16 Starter Set 1. This set includes one AIM16, one POW1, one ICON and one OCON. The AIM16 Starter Set 2 includes a MANMOD1 in place of the ICON. Both of these sets require that you have a hardware knowledge of your computer and of computer interfacing.

For simple plug compatible systems we also offer computer interfaces and sets for several home computers.

The PETMOD plugs into the back of the Commodore PET computer and provides two PET IEEE ports, one user port and one DAM SYSTEMS port. The PETMOD is connected to the AIM16 or XPANDR1 with CABLE A24. The PETSET1 includes one PETMOD, one CABLE A24, one AIM16, one POW1 and one MANMOD1. To read and display a single AIM16 channel (N) using the PETSET1 the BASIC statements

```
POKE59426,N:POKE59426,255:X=PEEK(59471):PRINT"CHANNEL "N"="X
```

are all that is needed.

The KIMMOD plugs into the COMMODORE KIM applications connector and provides one application connector and one DAM SYSTEM'S port. The KIMMOD is connected to the AIM16 or XPANDR1 with CABLE A24. Assembly and machine language programs for reading and displaying data are included. The KIMSET1 includes one KIMMOD, one CABLE A24, one AIM16, one POW1 and one MANMOD 1.

All sets come in American and European versions.

AIM16 Starter Set 1a (110 VAC) . . . $189.00
AIM16 Starter Set 1e (230 VAC) . . . $199.00
AIM16 Starter Set 2a (110 VAC) . . . $259.00
AIM16 Starter Set 2e (230 VAC) . . . $269.00
PETMOD . . . $ 49.95
KIMMOD . . . $ 39.95
PETSET 1a . . . $295.00
PETSET1e . . . $305.00
KIMSET1a . . . $285.00
KIMSET1e . . . $295.00

Coming Soon

TEMPSENS-2P with other temperature ranges. Interfaces for TRS-80, APPLE, AIM65. Light sensors. Output modules. Contact us for price and availability.

Dealers

Give your potential customers a reason for buying your computers. We offer excellent discounts to legitimate dealers. Contact us for our dealer pack.

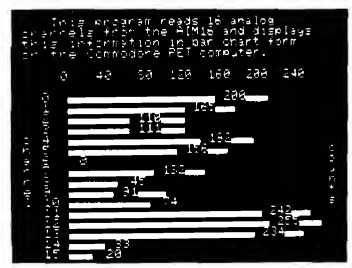
Our Guarantee of Satisfaction

Our customers are our most important asset. We want you to be pleased with whatever you purchase from us. We strive to offer top quality products at reasonable prices. We believe you should see an item before you spend your hard earned cash for it.

Ask for a demonstration at your local computer store so you can be sure our products perform as you want them to perform. Your dealer is a valued source of information and advice.

If you cannot see our products in advance, and order direct from us, we offer a money back guarantee. If our products don't perform as you expect, return the merchandise to us within 30 days, in its original condition, and we will refund the purchase price.

Our standard warranty for all our products is 90 days.



Order Form

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659   TWX: 710-456-0052

QUANTITY	DESCRIPTION	PRICE	TOTAL
SUBTOTAL			
Handling and shipping — add per order			$3.00
Foreign orders add 10% for AIR postage			
Connecticut residents add 7% sales tax			
TOTAL ENCLOSED			

NAME ____________________

COMPANY ____________________

ADDRESS ____________________

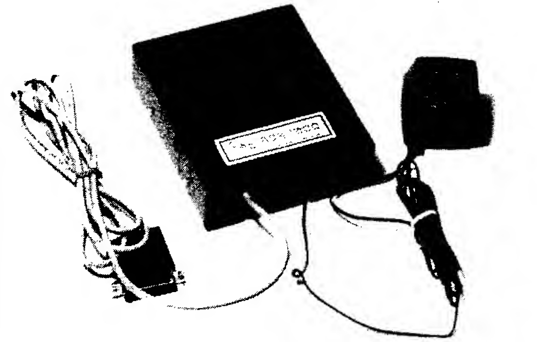
CITY ____________________

STATE ____________ ZIP ________

VISA ○ M/C ○ Expiration date ________

Card number ____________________

PET Printer Adapter



The CmC ADA 1200 drives an RS-232 printer from the PET IEEE-488 bus. Now, the PET owner can obtain hard copy listings and can type letters, manuscripts, mailing labels, tables of data, pictures, invoices, graphs, checks, needlepoint patterns, etc., using RS-232 standard printer or terminal.

A cassette tape is included with software for plots, formatting tables and screen dumps. The ADA1200 sells for $169.00 and includes case, power supply and cable.

Order direct or contact your local computer store.

VISA AND M/C ACCEPTED — SEND ACCOUNT NUMBER, EXPIRATION DATE AND SIGN ORDER.
ADD $3 PER ORDER FOR SHIPPING & HANDLING — FOREIGN ORDERS ADD 10% FOR AIR POSTAGE

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659   TWX: 710-456-0052

softside software

305 Riverside Drive New York, N.Y. 10025

the pet program.

1 GRAPHICS PAC 2
New Version

Quadruple your PET's *graphic resolution*. Why be stuck with the PET's cumbersome 25 x 40 1000 point display. With Graphics Pac you can directly control (set and clear) 4000 points on screen. It's great for *graphing, plotting, and gaming*. Graphics Pac allows you to plot in any combination of two modes: 4 Quadrant graphing with (0,0) center screen, and Standard graphing with (0,0) plotted in the upper left hand corner. Complete documentation shows how you can *merge* this useful routine with any of your own programs *without retyping* either one! All this on a high quality Microsette for only $9.95.

2 ASSEMBLER 2001

A *full featured assembler* for your PET microcomputer that follows the *standard set of 6502 mnemonics*. Now you can take full advantage of the computing abilities of your PET. *Store* and *load* via tape, *run* through the SYS or USR functions. *List* and *edit* too with this powerful assembler. No other commercial PETassembler gives you all these features plus the ability to *look at* the *PET'S secret Basic* ROMs *all in one program*. This valuable program is offered at $15.95

3 BIKE

An *exciting new simulation* that puts you in charge of a bicycle manufacturing empire. Juggle inflation, breakdowns, seasonal sales variations, inventory, workers, prices, machines, and ad campaigns to keep your enterprise in the black. *Bike is dangerously addictive*. Once you start a game you will not want to stop. To allow you to take short rest breaks, Bike lets you store the data from your game on a tape so you can continue where you left off next time you wish to play. Worth a million in fun, we'll offer BIKE at $9.95.

4 PINBALL

Dynamic usage of the PET's graphics features when combined with the fun of the *number 1 arcade game* equals an *action packed video spectacle* for your computer. Bumpers, chutes, flippers, free balls, gates, a jackpot, and a little luck guarantee a great game for all. $9.95

Authors: Our royalties are unbeatable

★ ☆☆☆☆☆☆☆ **MUSICAL MADDNESS** ☆☆☆☆☆☆☆ ★

SPECIAL — add an exciting new dimension to your PET computer with Soundware's soundsational music box and sonicsound software from Softside & Soundware — SOUND

☆THE SOUNDWORKS☆

The *Soundware music box* for your PET comes complete with controllable volume, an earphone jack, a *demo tape* with *two programs*, an instruction book, and a *one year warranty*. this sturdy unit is enclosed in an *attractive* plastic case. Notes tell how to *program your own sound effects*. All this during our *musical madness* for just 29.95

WORD FUN: *Speller*: fun ways to practice spelling + *Scramble* + *Flashcards* 9.95

☆MUSICAL SOFTWARE☆

ACTION PACK: *Breakthru* + *Target* + *Catterpillar*: non stop graphic action 9.95

PINBALL: a video action spectacle with real time flippers, chutes gates, bumpers, tags etc. 9.95

CLASSICS: *Checkers* + *Backgammon Board* +*Piano Player*; checkers vs. computer or friend. Piano plays Minute Waltz 9.95

MUSIC MANIA: Try to repeat a growing sequence of tones. With graphics. Challenge to the best ear 9.95

KIM • SYM • AIM • One magazine publishes more information about 6502 based systems, products and programs than all of the others combined: **• APPLE • PET • OSI**

MICRO™

P.O. Box 6502
Chelmsford, MA 01824

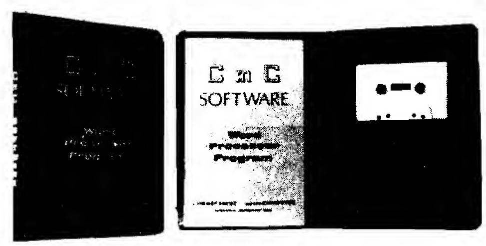
Are you tired of searching through computer magazines to find articles that relate to your 6502 system? MICRO magazine is devoted exclusively to 6502 systems. Each month, MICRO publishes application notes, hardware and software tutorials, interfacing information and program descriptions with complete source listings. MICRO is not just fun and games. It is the complete reference source for all 6502 enthusiasts.

You can order twelve issues of MICRO for $15.00, or for $18.00 outside the United States. Air mail subscriptions cost $27.00 in Central America, $33.00 in Europe and South America, and $39.00 in all other countries.

MICRO has been published regularly since October, 1977. Articles that appeared in the earlier issues of MICRO may be obtained in two bound anthologies, BEST OF MICRO Volume 1 and the companion collection ALL OF MICRO Volume 2, both available at computer stores or from the magazine.

PET Word Processor

8K and 16/32K PET versions



This program permits composing and printing letters, flyers, advertisements, manuscripts, etc., using the COMMODORE PET and a printer.

Printing directives include line length, line spacing, left margin, centering and skip. Edit commands allow you to insert lines, delete lines, move lines and paragraphs, change strings, save files onto and load files from cassette (can be modified for disk), move up, move down, print and type.

Added features for the 16/32K version include string search for editing, keyboard entry during printing for letter salutations, justification, multiple printing and more.

A thirty page instruction manual is included.

The CmC Word Processor Program for the 8K PET is $29.50. The 16/32K version is $39.50.

Order direct or contact your local computer store.

VISA AND M/C ACCEPTED — SEND ACCOUNT NUMBER, EXPIRATION DATE AND SIGN ORDER.
ADD $1 PER ORDER FOR SHIPPING & HANDLING — FOREIGN ORDERS ADD 10% FOR AIR POSTAGE

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659   TWX: 710-456-0052

SYM - 1 Baudot TTY Interface

Do not let the title fool you! This article has a lot more than just TTY stuff. Some of the techniques presented can be applied in many other situations.

Richard A. Leary
1363 Nathan Hale Drive
Phoenixville, PA 19460

One major shortcoming of the KIM is the inability to change the I/O routines without duplicating large parts of the monitor. In designing the SYM-1, Synertek nicely handled that shortcoming by vectoring all I/O calls through jumps located in SYSTEM RAM. Since those jumps are alterable by the user, almost any I/O device handler could be written and used with no effect on the rest of the monitor. That fact, coupled with the low cost of the Baudot Teletypes such as the Model 15 led me to develope SYM-1 I/O handlers for a 60 word per minute Model 15. Since the SYM-1 allows additional ROMs to be added to the board I placed these routines in an INTEL 2716 EPROM, along with some other system software. More on that later.

BAUDOT TTY

First, some background on Baudot Teletypes: A Baudot teletype uses only 5 data bits (unlike the 7 used by the ASCII Teletypes such as the Model 33) and thus it can only generate at most 32 unique code combinations. In order to expand that character set, two of the codes have been assigned special carriage shift functions much as it is done in a conventional typewriter. These two codes are called letters (LTRS) and figures (FIGS) and refer to the "lowercase" and "uppercase" character sets. Unlike a conventional typewriter, if a Baudot teletype is sent a LTRS code it stays in that mode until a FIGS code is sent. As a result of this shift method of operation, each key, except for some special ones, can send two different characters to another Model 15 TTY depending on the last shift sent. The receiver must of course remember what the last shift sent was, and print each succeeding key accordingly. While the Teletype does that remembering mechanically, it is obvious that a computer could easily do it electronically. That principle is the keystone of my software approach.

I mentioned that some keys or codes are assigned unique meanings regardless of whether a LTRS or FIGS code was the last code sent or received. In most Model 15 Teletypes those special keys are:

LTRS, FIGS, CAR RET (RETURN)
LINE FEED, SPACE, NULL (BLANK)

the net effect is that of the 32 code combinations, 26 have dual meanings. As a result, $2 \times 26 + 4 = 56$ unique characters can be printed on most Model 15 Teletypes. Note that I did not include LTRS and FIGS in the above total, as they are not really characters.

It should be obvious that with only 56 unique characters possible a 64, 96, or 128 character ASCII set cannot be directly generated or printed by a Baudot Teletype. The approach I used is an *indirect* approach which, much like the LTRS and FIGS codes, uses a sequence of codes to represent a character.

Hardware

A few points about Baudot Teletypes are appropriate before we begin. First, the electrical characteristics of a Model 15 are a bit different from those of a Model 33 ASCII Teletype. Rather than a 20mA current loop, a Model 15 usually has a 60mA current loop. Even more importantly, the supply voltage specified for that loop supply is usually 150v or more. Making a direct interface to the SYM-1 at those voltage and current levels would be disastrous. The answer to that problem is to use the conventional 20mA interface of the SYM-1, but to couple it to the Model 15 through opto-isolators. The opto-isolators will protect the SYM-1 and allow easy conversion of the signals to the Model 15 voltage in that the SYM-1 signal ground can remain isolated from the Model 15 ground. Since Model 15s are notoriously electronically noisy, the benefit of that isolation is that the probability of noise problems in the SYM-1 is sharply reduced.

The schematic of the interface I used and the power supply I built are shown in Figures 1 and 2. The only critical components in the interface are the selector magnet driver transistor and the zener diode. The voltage rating of the transistor must be high enough to withstand the open circuit loop supply voltage. The zener across that transistor must be similarily rated, as its function is to damp the magnet induced voltage spikes (positive and negative) and thus protect the driver transistor. The transformer in the power supply is not critical as long as it can supply 60mA continuously. This is a good opportunity to use one of the old "tube-type" power supply transformers that you probably have in you junk box.

The total resistance value of the series dropping resistors in the power supply may have to be altered if the open circuit voltage of your supply is higher than the approximately 190v put out by my supply. Once the supply is built, the variable resistor is adjusted to give a 60mA loop current when in the local mode and when no key is depressed.

Second, not all Model 15s are the same. There is a wide variety in code vs character terms, as well as in speed. At least three different speed Model 15s exist. For example, my machine was at one time a "weather" Teletype and had a special character set for most of the FIGS shift positions. I converted most of those keys and type elements to the "standard" communications set. I did leave in some characters which are not standard, and hence a few characters which I can print will have to be changed for a standard machine. Those characters are:

↑ – on standard
\+ " on standard
– Null on standard

The software changes required to accomodate the standard code are minor.

Finally, the Baudot machines have a few good and bad overall points which each user must consider before taking the plunge. They are:

a. Cheap: $50 - $100 or less should get you a good Model 15. Insist on a synchronous motor rather than a governor regulated motor.
b. Slow: 60 wpm translates to 45 Baud (6 char/sec) which is a little bit faster than one half the speed of a Model 33. The effective speed is even slower due to the necessity to send LTRS and FIGS shifts on an irregular basis.
c. Reliable: The Model 15 is probably 70% steel and 20% cast iron with a smattering of nonferrous materials. It just does not break if kept lubricated. (Remember—the Model 15 was the mainstay of the 24 hour per day news wire services.)
d. Heavy: All that iron!
e. Smelly: All that lubricant!
f. Repairable: If it does break, parts are available and the manuals are complete and explicit. I buy my parts from a company called Typetronics, Box8873, Fort Lauderdale, FL 33310. Prices are very reasonable and the response is nearly instantaneous.

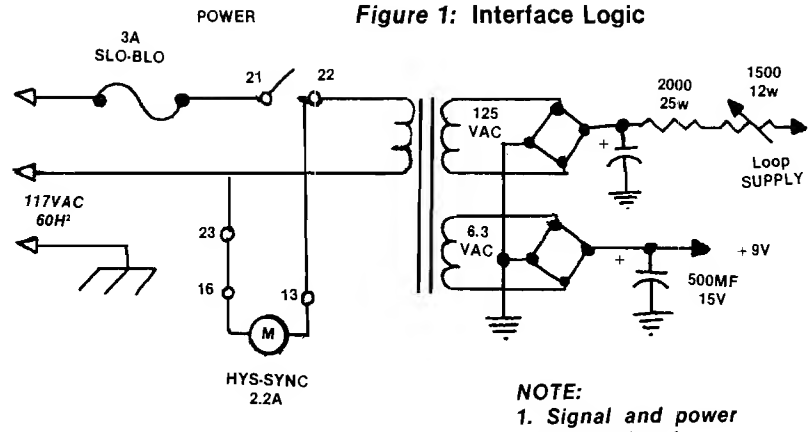


***Figure 1:* Interface Logic**

NOTE:
1. Signal and power ground isolated

The most difficult part of the hardware interface has already been described. Once the interface hardware is built it is connected to the standard current loop I/O pins on the SYM-1. The only SYM-1 hardware change required is a jumper change if you place the interface software in an EPROM and mount it on the SYM-1 as I did. Any of the empty ROM sockets can be used. Which one you use will depend upon whether you may have already installed the BASIC ROMs, the EDITOR/ASSEMBLER ROM, or some other device. I placed my 2716 in Socket U21 and since I had set the starting address of the software to $9000, I removed two jumpers and added a new one to enable U21 for $9000 to $97FF. Regardless of what you do, the SYM manual gives a complete description of the jumpers and how to set them for various ROM-Address combinations.

NOTES:
1. Resistors ¼w,5%
2. Open circles with numbers are TTY internal terminals.

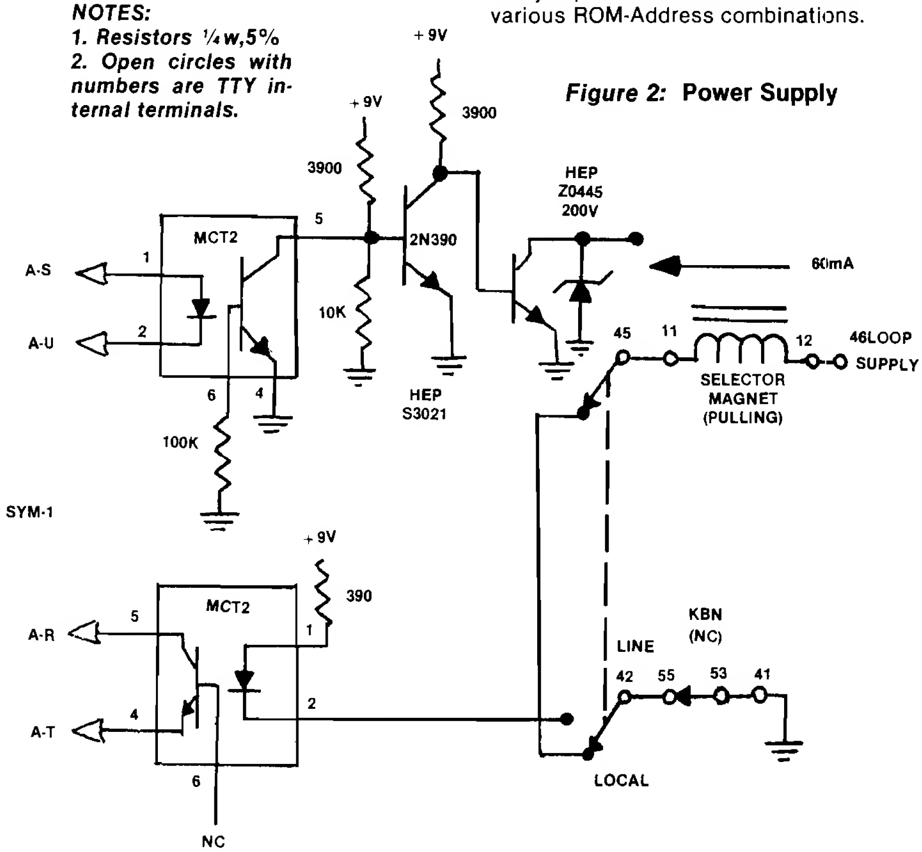


***Figure 2:* Power Supply**

Software

Given the character set limitations of the Model 15, the first software design requirement I had to address was how to represent the full ASCII character set when printing, and how to generate that set when using the keyboard. Let's look at the output case first.

The first decision was to convert all lower case alphabetic characters (a to z) to upper case. As the software will later show, that is a simple step. Moreover that approach does not create large problems in application, since the need for lower case alphabetic characters is limited unless one is doing word processing. If that is the case I doubt that the Model 15 is the answer in any case.

The second decision was to print those characters not normally printable by a Baudot Teletype as a four character sequence, beginning and ending with a period. For example, an equal sign ($3D) would be printed as the sequence .EQ.. In addition, of the 32 control codes which the ASCII can represent, I decided to recognize only RETURN, LINEFEED, BELL, and NULL. All others are ignored.

The decision as to how to generate the codes from the keyboard was a bit trickier. The approach selected was to use BELL as an escape character. This simply means that once the character is entered, the following entry or entries are handled in a special way until certain conditions are met. While any key could be used, BELL has the advantage of being seldom used as an input character and thus its use as an escape character is not a big loss. As will be seen later, it is not a loss at all as I can still generate a BELL as an ASCII character.

In the software which I created, the sequence following the escape character is only one or two key strokes in length. In fact it is only two when a LTRS must be entered between the BELL and the operative key. To better understand how this works, consider the following two examples which show both an escape sequence using a key, and the normal mode for that key:

	USER TYPES	ASCII OUTPUT TO SUPERMON
1	BELL	none
	=	= ($3D)
	:	: ($3A)
2	A	A ($41)
	FIGS	none
	BELL	none
	LTRS	none
	A	SOH ($01)

As the examples show, the BELL—(char) sequence generates the special character while the character by itself generates that character. The complete table of both input and output character translations is contained in the following table. LTRS and FIGS shifts are obviously required for some of the sequences shown, but have been omitted for brevity. Note that while lower case alphabetic characters cannot now be generated, it would be easy to add a double escape feature in order to do that.

TRANSLATION TABLE

ASCII	INPUT	OUTPUT	ASCII	INPUT	OUTPUT
NULL	NULL	NULL	'	'	'
SOH	BELL A		(	(	(
STX	BELL B		)	)	)
ETX	BELL C		*	BELL ↑	.AS.
EOT	BELL D		+	+	+
ENQ	BELL E		,	,	,
ACK	BELL F		-	-	-
BEL	BELL G	BELL	.	.	.
BS	BELL H		/	/	/
HT	BELL I		0	0	0
LF	LINEFEED	LINEFEED	1	1	1
VT	BELL K		2	2	2
FF	BELL L		3	3	3
CR	CAR RET	CAR RET	4	4	4
SO	BELL N		5	5	5
SI	BELL O		6	6	6
DLE	BELL P		7	7	7
DC1	BELL Q		8	8	8
DC2	BELL R		9	9	9
DC3	BELL S		:	:	:
DC4	BELL T		;	;	;
NAK	BELL U		<	BELL -	.LT.
SYN	BELL V		=	BELL :	.EQ.
ETB	BELL W		>	BELL +	.GT.
CAN	BELL X		?	?	?
EM	BELL Y		@	BELL &	.AT.
SUB	BELL Z		A to Z	A to Z	A to Z
ESC	BELL BELL		[	BELL (	.LB.
FS	BELL 1		\	BELL ;	.BS.
GS	BELL 2		]	BELL)	.RB.
RS	BELL 3		↑	↑	↑
US	BELL 4		_	BELL SPACE	.UN.
SPACE	SPACE	SPACE	`	BELL $	.AG.
!	!	!	a to z		A to Z
"	BELL '	.QT.	{	BELL .	.LP.
#	#	#	|	BELL !	.VS.
$	$	$	}	BELL ;	.RP.
%	BELL /	.PC.	~	BELL #	.TL.
&	&	&	DEL	BELL NULL	.DL.

Software Implementation

I wish that I could report that the software is very compact and that it uses a great deal of the SUPERMON routines. While the software is not large, at about 1/2K it is not small. On the other hand, the only SUPERMON routines I use are:

- SAVER @ $8188
- RESXAF @ $81B8
- RESALL @ $81C4
- DLYH @ $8AE9

Also, I use the following RAM and SYSTEM I/O locations:

- CHAR @$59 used like SUPERMON does,
- TTYMDE @ $100 current shift position,
- PBDA @ $A402 I/O port,
- PBDA + 1 I/O port direction resister,
- SDBYT @ $A651 timing constant,

As the complete listings indicate, the two top level routines for input and output are ASCIN and ASCOUT respectively. Each of these routines calls other routines to do the actual input with the Teletype. The functions of each routine and its general characteristics are summarized in the following chart.

ROUTINE	FUNCTION	INPUT	OUTPUT	ALTERS	CALLS
ASCIN	get input for full char set using escape sequence	none	ASCII char in A	A,F	SAVER CHRIN RESXAF
ASCOUT	print special sequences or direct char to CHROUT for output	ASCII char in A	none	none	SAVER CHROUT RESALL ALTOUT
CHRIN	get key,echo and convert to ASCII	(TTY key)	ASCII char in A	A,F TTYMDE CHAR	SAVER HALF FULL TTYOUT
CHROUT	convert ASCII char to Baudot and handle mode changes	ASCII char in A	none	TTYMDE	SAVER TTYOUT RESALL
TTYOUT	output Baudot char	Baudot char in A	(TTY tape)	PBDA PBDA+1 A,F,X Y,TTYMDE	FULL
HALF	delay 11ms	none	none	X	DLYH
FULL	delay 22ms	none	none	X	DLYH

Note that SDBYT must be set to $2D before using these routines as that parameter is used by DLYH as a timing constant. For teletypes running at higher speeds, either the value of SDBYT or the loop values used in HALF and FULL should be reduced. If your machine is running slightly faster or slower than mine the input or output may not be completely reliable. If that is the case, adjust SDBYT up or down as appropriate, until all functions work error free.

In my system the first routine in software is used to set SDBYT and alter INVEC and OUTVEC to point to the Baudot routines.That is not absolutely necessary, but does make the start-up process somewhat easier. All I have to do is enter G9000 CR on the SYM-1 keyboard and the Baudot I/O package is up and running.

One point of emphasis—as written, this software includes an internal echo for each input. The input sequence is echoed literally. This means that if an escape sequence of BELL / is entered, what is echoed is precisely that and not the .PC. which would be output if the SYM output a %. It would of course be possible to change that approach. A translated echo would be a bit slower since each escape sequence would be echoed as six or seven characters due to the .xx. sequence and the necessary shifts.

Conclusion

I hope that you find this package useful. If any questions need answering, please feel free to contact me. And if anyone would like the code translations changed I would be glad to reassemble the software and provide the revised program in listing form for the cost of the materials and postage. Good Luck.

```
0000           ;
0000           ;SYM-1 BAUDOT TTY I/O PACKAGE
0000           ;
0000           ;Fixed Parameters
0000           ;
0000           NULL   =0
0000           FIGMDE =0
0000           BELL   =7
0000           ESC    =$1B
0000           FIGS   =$1B
0000           LTRS   =$1F
0000           SPACE  =$20
0000           LTRMDE =$20
0000           DELETE =$7F
0000           ;
0000           ;System RAM Assignments
0000           ;
0000           CHRBUF =$F9            character buffer
0000           TTYMDE =$100           TTY shift mode
0000           SDBYT  =$A651          timing constant
0000           INVEC  =$A660          input jump vector
0000           OUTVEC =$A663          output jump vector
0000           ;
0000           ;SUPERMON Routines
0000           ;
0000           SAVER  =$8188          register save
0000           RESXAF =$81B8          restore except A&F
0000           RESALL =$81C4          restore registers
0000           DLYH   =$8AE9          delay
0000           ACCESS =$8B86          un-protect system RAM
0000           ;
0000           ;I/O Devices
0000           ;
0000           PBDA   =$A402          TTY port
0000           ;
0000           ;I/O VECTOR INITIALIZATION
0000           ;
9000                  *=$9000
```

```
9000 20868B  BEGIN  JSR ACCESS          un-protect sys ram
9003 A920           LDA #$2D            change timing
9005 8D51A6         STA SDBYT           in SUPERMON
9008 A985           LDA #ASCOUT*256/256 point output
900A 8D64A6         STA OUTVEC+1          to ASCOUT
900D A990           LDA #ASCOUT/256       TTY handle
900F 8D65A6         STA OUTVEC+2          output line
9012 A91D           LDA #ASCIN*256/256    then point
9014 8D61A6         STA INVEC+1           input to
9017 A990           LDA #ASCIN/256        BAUDOT TTY
9019 8D62A6         STA INVEC+2           input handle
901C 60             RTS                 then return
901D                ;
901D                ;ASCII Input
901D                ; Uses BELL as escape character to
901D                ; generate full ASCII character set
901D                ; except for lower case alpha.
901D                ;
901D 208881  ASCIN  JSR SAVER           save registers
9020 20EB90         JSR CHRIN           get char
9023 C907           CMP #BELL           if not BELL
9025 D01B           BNE EXTAIN          then return
9027 20EB90  ESCAPE JSR CHRIN           get second char
902A C907           CMP #BELL           if not BELL
902C D004           BNE NOTESC          then jump
902E A91B           LDA #ESC            get ESC code
9030 D010           BNE EXTAIN          and return
9032 C900    NOTESC CMP #NULL           if not NULL
9034 D004           BNE NOTDEL          then jump
9036 A97F           LDA #DELETE         else get DEL
9038 D008           BNE EXTAIN          and return
903A C520    NOTDEL CMP SPACE           if less than space
903C 90E9           BCC ESCAPE          try again
903E AA             TAX                 move char to index
903F BD4590         LDA ASCII,X         and get translation
9042 4CB881  EXTAIN JMP RESXAF          restore and return
9045 5F      ASCII  .BYTE $5F,$7C,0,$7E,$60,0,$40,$22
9046 7C
```

```
9047 00
9048 7E
9049 60
904A 00
904B 40
904C 22
904D 5B            .BYTE $5B,$5D,0,$3E,$7D,$3C,$7B,$25
904E 5D
904F 00
9050 3E
9051 7D
9052 3C
9053 7B
9054 25
9055 00            .BYTE 0,$1C,$1D,$1E,$1F,0,0,0
9056 1C
9057 1D
9058 1E
9059 1F
905A 00
905B 00
905C 00
905D 00            .BYTE 0,0,$3D,$5C,0,0,0,0
905E 00
905F 3D
9060 5C
9061 00
9062 00
9063 00
9064 00
9065 00            .BYTE 0,1,2,3,4,5,6,7
9066 01
9067 02
9068 03
9069 04
906A 05
906B 06
906C 07
906D 08            .BYTE 8,9,$A,$B,$C,$D,$E,$F
906E 09
906F 0A
9070 0B
9071 0C
9072 0D
9073 0E
9074 0F
9075 10            .BYTE $10,$11,$12,$13,$14,$15,$16,$17
9076 11
9077 12
9078 13
9079 14
907A 15
907B 16
907C 17
907D 18            .BYTE $18,$19,$1A,0,0,0,$2A,0
907E 19
907F 1A
9080 00
9081 00
9082 00
9083 2A
9084 00
9085               ;
9085               ;ASCII Output
9085               ; Outputs characters normal - not
9085               ; printed by BAUDOT TTY as [illegible]
9085               ; character sequence of the [illegible]
9085               ; .xx. All other characters are
9085               ; forwarded to CHROUT for printing.
9085               ;
9085 208881  ASCOUT JSR SAVER          save registers
9088 297F           AND #$7F           mask out msb
908A A230           LDX #LSTSQ-MULTI+256/256 get offset
908C DDB390  TSTCHR CMP MULTI,X        test char
908F F00A           BEQ OUTSTR         if match send seq
9091 A003           LDY #3             set up to
9093 CA      MVPNTR DEX                count down
9094 301A           BMI NTFND          but if no match jump
9096 88             DEY                count down
9097 D0FA           BNE MVPNTR         until five less
9099 F0F1           BEQ TSTCHR         then go test next
909B A002    OUTSTR LDY #2             send 2 char
909D 20E690         JSR PRDOUT         send period
90A0 E8      NXTOUT INX                move pointer
90A1 BDB390         LDA MULTI,X        and get char
90A4 204F91         JSR CHROUT         and send it
90A7 88             DEY                loop until
90A8 D0F6           BNE NXTOUT         all sent
90AA 20E690         JSR PRDOUT         send period
90AD 4C0481         JMP RESALL         then return
90B0 4C5491  NTFND  JMP ALTOUT         jump for single char
90B3 22      MULTI  .BYTE '"QT'
90B4 51
90B5 54
90B6 25             .BYTE '%PC'
90B7 50
90B8 43
90B9 2A             .BYTE '*AS'
90BA 41
90BB 53
90BC 3C             .BYTE '<LT'
90BD 4C
90BE 54
90BF 3D             .BYTE '=EQ'
90C0 45
90C1 51
90C2 3E             .BYTE '>GT'
90C3 47
90C4 54
90C5 40             .BYTE '@AT'
90C6 41
90C7 54
90C8 5B             .BYTE '[LB'
90C9 4C
90CA 42
90CB 5C             .BYTE '\BS'
90CC 42
90CD 53
90CE 5D             .BYTE ']RB'
90CF 52
90D0 42
90D1 5F             .BYTE '_UN'
90D2 55
90D3 4E
90D4 60             .BYTE '`AG'
90D5 41
90D6 47
90D7 7B             .BYTE '{LP'
90D8 4C
90D9 50
90DA 7C             .BYTE '|VS'
90DB 56
90DC 53
90DD 7D             .BYTE '}RP'
90DE 52
90DF 50
90E0 7E             .BYTE '~TL'
90E1 54
90E2 4C
90E3 7F      LSTSQ  .BYTE DELETE
90E4 44             .BYTE 'DL'
90E5 4C
90E6                ;
90E6                ;Output Period
90E6                ;
90E6 A92E    PRDOUT LDA #'.            get period
90E8 4C4F91         JMP CHROUT         go do it
90EB                ;
90EB                ;Character Input
90EB                ; Gets input from BAUDOT keyboard
90EB                ; and converts to ASCII and echos
90EB                ; character.
90EB                ;
90EB 208881  CHRIN  JSR SAVER          save registers
90EE A900    AGAIN  LDA #NULL          clear buffer
90F0 85F9           STA CHRBUF         in RAM
90F2 2C02A4  LOOK   BIT PBDA           test for start
90F5 50FB           BVC LOOK           if not loop
90F7 200292         JSR HALF           else wait half bit
90FA 2C02A4         BIT PBDA           and test again
90FD 50F3           BVC LOOK           if false start over
90FF A007           LDY #7             get seven bits
9101 200692  NXTIN  JSR FULL           wait bit time
9104 2C02A4         BIT PBDA           test input
9107 18             CLC                0 if no ovflw
9108 5001           BVC SAVE           save if 0
910A 38             SEC                else is 1
910B 66F9    SAVE   ROR CHRBUF         shift into buffer
910D 88             DEY                count down
910E D0F1           BNE NXTIN          loop if more
9110 A5F9           LDA CHRBUF         get char
9112 4A             LSR A              and finish shift
9113 49FF           EOR #$FF           complement
9115 291F           AND #$1F           get 5 data bits
9117 48             PHA                save char
9118 20E091         JSR TTYOUT         then echo
911B 68             PLA                restore char
911C C91F           CMP #LTRS          if not LTRS
911E D008           BNE NOTLTR         then jump
9120 A920           LDA #LTRMDE        get LTRS mode code
9122 8D0001  SETMDE STA TTYMDE         and set mode
9125 4CEE90         JMP AGAIN          then try again
9128 C91B    NOTLTR CMP #FIGS          if not FIGS
912A D004           BNE NOTFIG         then jump
```

```
912C A900          LDA #FIGMDE  set FIGS mode code
912E F0F2          BEQ SETMDE   and set
9130 18     NOTFIG CLC          clear carry
9131 6D0001        ADC TTYMDE   convert to table
9134 85F9   TRYOTH STA CHRBUF   and save
9136 A25F          LDX #95      number char - 1
9138 BD8091 SEARCH LDA BAUDOT,X get table entry
913B 293F          AND #$3F     look at mode+data
913D C5F9          CMP CHRBUF   if same as buffer
913F F00A          BEQ FOUND    then found
9141 CA            DEX          else count down
9142 10F4          BPL SEARCH   and loop until all tested
9144 A5F9          LDA CHRBUF   get char
9146 4920          EOR #LTRMDE  complement mode
9148 4C3491        JMP TRYOTH   and try again
914B 8A     FOUND  TXA          move ASCII to A
914C 4C8881        JMP RESXAF   and return
914F        ;
914F        ;Output Single ASCII Character
914F        ; Converts to BAUDOT and handles
914F        ; mode changes as required.
914F        ;
914F 208881 CHROUT JSR SAVER        save registers
9152 297F          AND #$7F         mask out msb
9154 C960   ALTOUT CMP #$60         if upper case
9156 9002          BCC UPRCSE       skip convert
9158 29DF          AND #$DF         else make upper
915A AA     UPRCSE TAX              make char pointer
915B BD8091        LDA BAUDOT,X     get BAUDOT
915E C980          CMP #$80         if msb=1
9160 B01B          BCS NOPRNT       do not print
9162 C940          CMP #$40         if next bit=0
9164 9014          BCC NOMDE        no mode change
9166 48            PHA              else save char
9167 2920          AND #LTRMDE      look at mode bit
9169 CD0001        CMP TTYMDE       if same
916C F00B          BEQ NOCHNG       then no change
916E 8D0001        STA TTYMDE       else save
9171 4A            LSR A            move to
9172 4A            LSR A            correct
9173 4A            LSR A            position
9174 091B          ORA #FIGS        convert to char
9176 20E091        JSR TTYOUT       and send
9179 68     NOCHNG PLA              get char back
917A 20E091 NOMDE  JSR TTYOUT       send it
917D 4CC481 NOPRNT JMP RESALL       and return
9180 60   BAUDOT .BYTE $60,$FF,$FF,$FF,$FF,$FF,$FF,$45
9181 FF
9182 FF
9183 FF
9184 FF
9185 FF
9186 FF
9187 45
9188 FF          .BYTE $FF,$FF,2,$FF,$FF,8,$FF,$FF
9189 FF
918A 02
918B FF
918C FF
918D 08
918E FF
918F FF
9190 FF          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
9191 FF
9192 FF
9193 FF
9194 FF
9195 FF
9196 FF
9197 FF
9198 FF          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
9199 FF
919A FF
919B FF
919C FF
919D FF
919E FF
919F FF
91A0 04          .BYTE 4,$4D,$FF,$54,$49,$FF,$5A,$4B
91A1 4D
91A2 FF
91A3 54
91A4 49
91A5 FF
91A6 5A
91A7 4B
91A8 4F          .BYTE $4F,$52,$FF,$51,$4C,$40,$50,$5D
91A9 52
91AA FF
91AB 51
91AC 4C
91AD 40
91AE 5C
91AF 5D
91B0 56          .BYTE $56,$57,$53,$41,$4A,$50,$55,$47
91B1 57
91B2 53
91B3 41
91B4 4A
91B5 50
91B6 55
91B7 47
91B8 46          .BYTE $46,$58,$4E,$5E,$FF,$FF,$FF,$59
91B9 58
91BA 4E
91BB 5E
91BC FF
91BD FF
91BE FF
91BF 59
91C0 FF          .BYTE $FF,$63,$79,$6E,$69,$61,$6D,$7A
91C1 63
91C2 79
91C3 6E
91C4 69
91C5 61
91C6 6D
91C7 7A
91C8 74          .BYTE $74,$66,$6B,$6F,$72,$7C,$6C,$78
91C9 66
91CA 6B
91CB 6F
91CC 72
91CD 7C
91CE 6C
91CF 78
91D0 76          .BYTE $76,$77,$6A,$65,$70,$67,$7E,$73
91D1 77
91D2 6A
91D3 65
91D4 70
91D5 67
91D6 7E
91D7 73
91D8 7D          .BYTE $7D,$75,$71,$FF,$FF,$FF,$43,$FF
91D9 75
91DA 71
91DB FF
91DC FF
91DD FF
91DE 43
91DF FF
91E0        ;
91E0        ;BAUDOT Output
91E0        ;
91E0 A220   TTYOUT LDX #$20       set port to
91E2 8E03A4        STX PBDA+1     output mode
91E5 291F          AND #$1F       look at data bits
91E7 0960          ORA #$60       add 2 stop bits
91E9 49FF          EOR #$FF       complement
91EB 38            SEC            set start bit
91EC 2A            ROL A          get ready
91ED A008          LDY #8         setup for 8 bits
91EF 4A     NXTBIT LSR A          move lsb to C
91F0 48            PHA            save char
91F1 A900          LDA #0         clear for zero
91F3 9002          BCC OUTONE     if no carry jump
91F5 0920          ORA #$20       else set bit
91F7 8D02A4 OUTONE STA PBDA       send to port
91FA 68            PLA            get char back
91FB 200692        JSR FULL       delay one bit
91FE 88            DEY            then count down
91FF D0EE          BNE NXTBIT     if more then loop
9201 60            RTS            else quit
9202        ;
9202        ;Bit Timing Routines
9202        ;
9202 A20B   HALF   LDX #11        half period delay
9204 D002          BNE TIMLOP     so do it
9206 A216   FULL   LDX #22        full period
9208 20E98A TIMLOP JSR DLYH       delay 1ms
920B 48            PHA            fill
920C 68            PLA
920D CA            DEX            count down
920E D0F8          BNE TIMLOP     and loop
9210 60            RTS            until done
9211               .END
```

The MICRO Software Catalog: XIV

Mike Rowe
P.O. Box 6502
Chelmsford, MA 01824

Software Catalog Note

This regular feature of MICRO is provided both as a service to our readers and as a service to the 6502 industry which is working hard to develop new and better software products for the 6502 based system. There is no charge for listings in this catalog. All that is required is that material for the listing be submitted in the listing format. All info should be included. We reserve the right to edit and/or reject any submission. Some of the submissions are starting to get much too long. We might not edit the description the same way you would, so please, be brief and specific.

Name: **Environment for KIM BASIC**
System: **KIM running Microsoft BASIC**
Memory: **1.2K**
Language: **Machine language**
Hardware: any **KIM** that runs BASIC

Description: This software package provides the following utility programs for use with KIM BASIC: Renumber, Range Deletion, Append, Character-Oriented Line Editing, Automatic Line Number Prompting, Controlled Listings. The package is configured to interface itself automatically with any version of 9-digit KIM BASIC upon execution. There are no restrictions on length of internal references in lines; you can renumber from 1,2,3, to 63000,63010 and back again. Renumbers typical 200 line program in less than 10 seconds. Range deletions (i.e. Delete 100-950) take approxomately 5 seconds per 100 lines deleted. One POKE makes the next LOAD an APPEND and then restores regular LOAD status. All functions have complete error checks before changing your original program and report errors using BASIC's own error messages. Page length can be varied during listing or command mode at any point. Edit mode allows moving lines in the program or changing one section of a line without retyping the complete commented source listing.

Includes: KIM format tape, source, manual
Price: **$20.00** plus $1.50 shipping and handling. California residents add 6% sales tax.
Author: **Sean McKenna**
Available from:
Sean McKenna
64 Fairview Ave.
Piedmont, CA 94610

Name: **MEM-EXPLORER**
System: **Commodore PET**
Memory: **8K or more**
Language: **Microsoft PET BASIC** with 6502 machine-language subroutine
Hardware: **PET 2001-8, 2001-16, or 2001-32**

Description: MEM-EXPLORER gives the PET owner a "window" into his computer, to give an understandable view of memory contents—both user (RAM) and Interpreter/OS (ROM). When the program is run, you are asked for a starting location. MEM-EXPLORER then presents information on 20 bytes of memory, starting with the location you specified. In the left column is the address of the byte, while columns to the right hold the decimal value of its contents, the character equivalent (or BASIC token, if appropriate), and two different two-byte values (address, integer). By specifying the area in RAM where the BASIC program is stored, you can actually see the program "listed" vertically in the character column, and tell exactly where every character or token is stored. MEM-EXPLORER includes routines that allow it to be combined with your programs automatically.

Copies: **Many**
Price: **$7.95** (quantity discount available)
Includes: Cassette in Norelco-style box, description, operating instructions, and zip-lock protective package.
Designer: **Roy Busdiecker**
Available from: Better computer stores, or directly from
Micro Software Systems
P.O. Box 1442
Woodbridge, VA 22193

Name:	**Space Shuttle Landing Simulator**
System:	**APPLE II**
Memory:	**48K**
Language:	**Assembly and Applesoft II**
Hardware:	**6HIRES color APPLE and Applesoft II ROM card**

Description: Modeled after the real Shuttle Mission Simulator, this program is a real flight simulator. The HIRES screen shows the "out-the-window" view using animation, projective geometry, and high speed assembly language graphics to display the image of the runway, sky, mountain, clouds, etc. In text below the screen is the flight data plus warnings and messages. Real flight algorithms are tailored to the Shuttle Orbiter's flight characteristics providing realistic stick response using the game paddle. Functional features are: full stall capability, ejection, landing gear, speed brakes,and wheel brakes on roll out. Runway stripes on roll out give a speed indication. The instruction manual is 10 pages, over 3500 words, and provides a brief introduction to guiding flight.

Copies:	**Just Released** (20 Aug 79).
Price:	**$15.00 ppd.** New Mexico residents add 4% sales tax.
Author:	**John Martellaro**
Available from:	

Harvey's Space Ship Repair
P.O. Box 3478
University Park
Las Cruces, NM 88003

Name:	**XMON, an extended monitor for TIM**
System:	**any version of TIM**
Memory:	**minimum 512 bytes**
Language:	**6502 assembly**
Hardware:	**Mimimum TIM plus 2708 addressing and comparitor with 5 discretes; optional LED and 2 discretes.**

Description: Nine commands from terminal provide: fill memory with constant; move, compare memory blocks; search for string; go execute with breakpoint and single step trace; exit to TIM monitor; load and dump KIM format cassette at 4K/min. All functions externally callable; suitable for calling by TINY USR function. Standard version resides EC00 through EFFF.

Copies:	**Just Released**
Price:	**$28.00** for standard version, Add **$7.00** for relocation.
Includes:	2708 PROM, comparitor and discretes, instructions, schematics.
Author:	**Phil Lange**
Available from:	

206 Santa Clara Ave.
Dayton, Ohio 45405
(513) 278-0506

Name:	**APPLE II Sweet 16 Assembler**
System:	**APPLE II**
Memory:	**16K RAM, Cassette Deck**
Language:	**Machine and Sweet 16**

Description: This system is a co-resident, two pass assembler for Sweet 16, the 16 bit software processor resident in the APPLE II Rom. The assembler has full cursor editing capabilities identical to those of Applesoft, English Language error messages, and line length up to 255 characters for extended program documentation. Commands are included to read and write the text file to tape, display the input format, renumber lines, list text file, return to APPLE monitor, and to assemble. The assembler supports pseudo OPS to determine ASCII strings, define hex strings, label location, and define program origin. The assembler lists addresses, object code, source code and symbol table. Included with the program is full documentation for use of the assembler, plus a full description of all Sweet 16 OP codes and16 bit registers and short programs illustrating each operation.

Copies:	**Just Released**
Price:	**$15.00**
Author:	**Steve Cochard**
Available from:	

Scientific Software
P.O. Box 156
Stowe, PA 19464

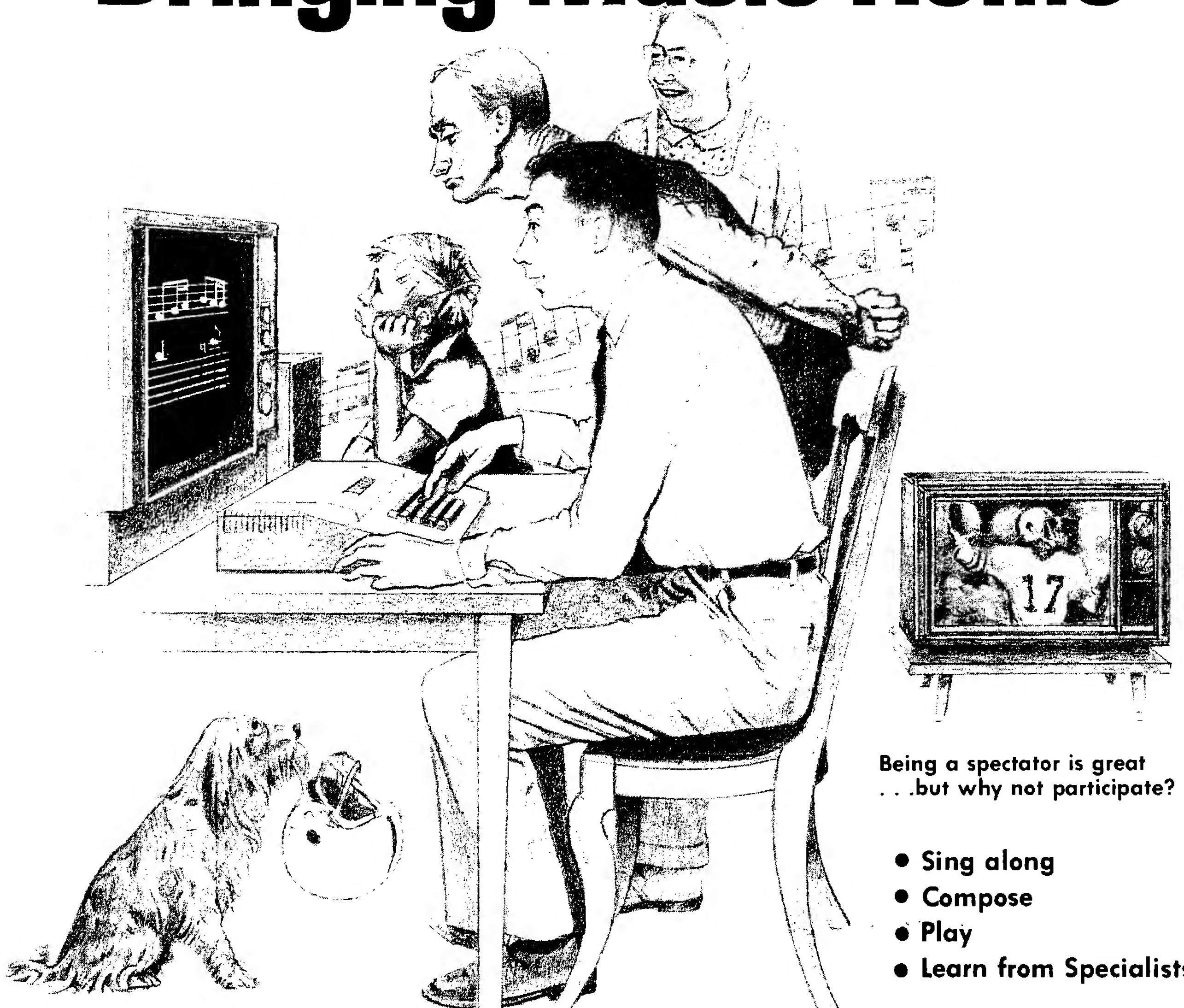
Name:	**AMPER-SORT II**
System:	**APPLE**
Memory:	**32K minimum**
Language:	**Assembler**

Description: AMPER-SORT II is an enhanced version of the AMPER-SORT routine published in MICRO, number 14. Two major enhancements improve sort speed and increase its versatility. The Shell-Metzner algorithm reduces sort time and a capability to sort two-dimensional character string arrays enables AMPER-SORT II to be used easily with programs such as FILE CABINET, an Apple Contributed Software Bank program. FILE CABINET with AMPER-SORT II will sort 100 records of 3 10-byte-average fields in 3 seconds compared to 7 minutes using the original BASIC sort code. AMPER-SORT II will sort integer arrays, floating point arrays, and one or two-dimensional string arrays. It also features an easy-to-use BASIC interface to pass array name and sort parameters.

Copies:	**Just Released**
Price:	**$15.95.** (California residents add 6% sales tax)
Author:	**Alan G. Hill**
Available from:	

PROGRAMMA INTERNATIONAL, Inc.
3400 Wilshire Blvd.
Los Angeles, CA 90010

Bringing Music Home



Being a spectator is great
. . .but why not participate?

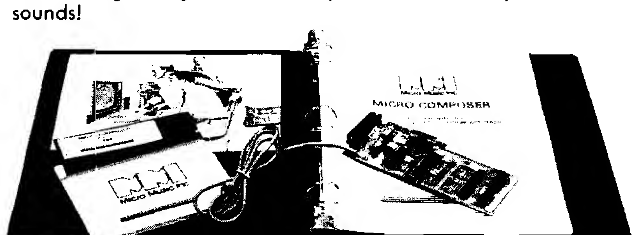
- **Sing along**
- **Compose**
- **Play**
- **Learn from Specialists**

LET MICRO MUSIC TURN YOUR APPLE II ® INTO A FAMILY MUSIC CENTER!

VISIT THE APPLE DEALER NEAREST YOU AND ASK FOR A DEMONSTRATION OF MMI'S MICRO COMPOSER ™
The MICRO COMPOSER LETS YOU—

- **Play** up to 4 **simultaneous voices**
- See all **4 voices** at the same time you're **hearing** the music—a must for music editing!
- **Enter** music notes by a fast, simple and well-tested coding system.
- **Program** the **pitch, rhythm,** and **timbre of the music. Tempo** is varied by the Apple paddle.
- **Choose 7 different tone colors** for each voice or **create your own tone color.**
- **Compose, edit, display,** and **play** music through an interactive, command-driven language that's easy to learn.
- **Save** your **music** on disk or cassette.
- **Hear** quality music sound at low cost through the **MICRO MUSIC ™ DAC** card. No amplifier needed! Designed for MMI by Hal Chamberlin and Micro Technology Unlimited.
- **Select** from future MMI music instruction software to accompany the **MICRO MUSIC DAC.**

The **MICRO COMPOSER** is an **APPLE II®** compatibile, low-cost music system designed by the folks at MMI. Our music software was designed by leading experts in music education. A simple step-by-step instruction manual leads you through entering, displaying, editing, and playing music with up to four voices—soprano, alto, tenor, and bass. You can change the sound of each voice to reed, brass, string, or organ sounds and you can even color your own music sounds!



HAVE FUN! THE MICRO COMPOSER comes complete with an instruction manual, software disk or cassette—in either Integer or Applesoft **ROM BASIC,** and the **MICRO MUSIC DAC** music card. Just plug the **MICRO MUSIC DAC** into the **APPLE** extension slot and connect the audio cable to a speaker.

Suggested retail price $220.

Ask your local dealer for information on MMI products, or contact:

MMI ™
Micro Music Inc 309 Beaufort, University Plaza, Normal, IL 61761

APPLE II is a trademark of Apple Computer Inc

MICRO Reviewer

One of the most common requests I receive from our readers is that MICRO provide reviews of hardware and software products. One of the most common types of articles MICRO receives is the product review. Why then, you may reasonably ask, hasn't MICRO printed lots of reviews? The answer is simple. While some other magazines print product reviews as filler material, I feel that a product review is a very special trust and must be handled in special ways. I feel that any review printed in MICRO should be as accurate, unbiased, and complete as possible, and that the qualifications and potential "conflicts of interest" of the author should be known.

Unsolicited Reviews

Think for a moment about the unsolicited reviews MICRO receives. Why did the author write the review? Probably for one of two reasons:

He loved the product, or,
He hated the product.

In either case the author is biased. An even more serious problem with the unsolicited review is that an author could have a vested interest in a product. He might be a friend of the manufacturer of the product, or could even be the manufacturer himself! Another problem is that the coverage of the possible products is going to be very spotty. Since every author is free to choose what he is going to review, some very good products will be overlooked, and some bad ones, too.

A Plan

I have come up with a plan which I feel will permit MICRO to obtain the types of reviews it wants and which the readers require. Phase 1 of the plan entails getting a list of qualified, unbiased reviewers. This panel of reviewers would each fill out the attached form and submit it to MICRO. Authors would then be selected from this group to review products. Since the form provides a means by which the basic qualifications of the authors may be determined, and since the selection would be made by MICRO, not the individual authors, both the qualification and bias problems should be solved.

Reviewer Qualification Form

Why should you become a reviewer for MICRO? I can think of a number of good reasons. First, you will get a chance to try new products, often before they become generally available. Second, you will get a chance to help fellow computerists by providing the detailed information they need to help decide on the merits of various products. Third, you will have your review published under your "byline". Fourth, you will be paid by MICRO for the review. Fifth, you will normally be able to purchase the product you reviewed at a substantial discount.

If you would like to become a reviewer for MICRO, please complete and return the attached form.

Robert M. Tripp

MICRO Reviewer Qualification Form

Name: ....

Address: ....

City: .... State: .... Zip: ....

Phone: Days: .... Evenings: ....

List all 6502 Hardware you own or have 'unlimited' access to: ....

....

List Programming languages you are qualified in and can use on your equipment: ....

....

List types of 6502 applications you are interested in: ....

....

List any special equipment which your system uses/supports: ....

....

List **ALL** companies, stores, etc. with which you have any relationship other than as customer, and **ANY** company, store, etc. whose products you do not feel you should review for any other reason:

....

Write a brief biography which may be published with your reviews which gives the reader a summary of your interests and qualifications: ....

....

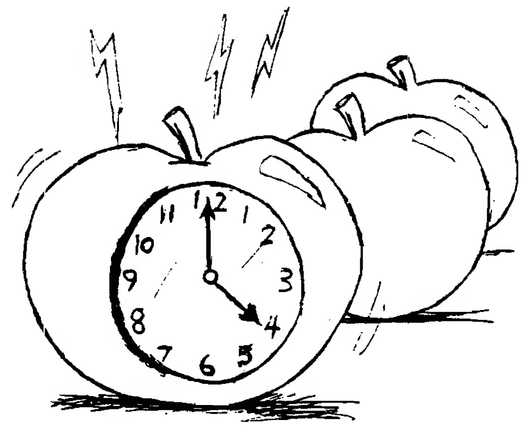
....

I declare that the above information is complete and accurate.

Signed: .... Date: ....

Please complete, using additional pages if necessary, and return to: **MICRO, P.O. Box 6502, Chelmsford, MA 01824**

Alarming APPLE



Paul Irwin
P.O.B. 1264, Station B
Ottawa, Ontario K1P 5R3
Canada

Here is a way to program you APPLE to respond to errors with an alarm and keyboard lockout.

Instead of using the CTRL-G beep on your next program, here's an alarm system written to assist in performing error recovery on the APPLE II. When the alarm system is used, your program will react to an error by immediately locking the keyboard, sounding a continuous two-tone alarm, and forcing the operator's attention to an error recovery subroutine. No way will recognizable errors escape your edits once they meet the Alarming APPLE!

To use the alarm system, start with each of your subroutines clearly defined as either *error detecting* or *error correcting*. This means that you will classify most of your "normal" routines as error detecting routines. Arrange to have all of your routines invoked by a mainline. Then the mainline can invoke error correcting routines, as well, and still remain in control. This is illustrated by the program shown here.

In the BASIC listing, the one error detecting routine is called TASK, while the error correcting routine is TRAP. The mainline is free to decide what to do after recovery: whether to continue the same error detecting routine or to take any other action. An intelligent mainline of this sort can avoid most error recovery hassles.

The key to the error recovery procedure is a machine language routine called ALARM. It is invoked from BASIC by executing a CALL 3529 and from machine language by executing a JSR $DC9. The alarm routine will then generate a two-tone alarm continuously. At the end of each cycle, it examines the keyboard for a CTRL-C. If none was found, it continues sounding the alarm. But when a CTRL-C is typed, the sound will stop and the routine will return. The effect is to produce a continuous sound, ignoring any input, until a CTRL-C is entered.

You may have your own ideas as to how the alarm should sound. The duration of the first tone is in $DA2 and its period is in $D9D. The second tone has its duration and pitch stored in $DBF and $DBA. The two that I employ are quite noisy, but you can experiment with other parameter pairs. Those periods that are relatively prime — having no common factor — will produce discord. They will be loudest when matching the APPLE's speaker resonance.

When loading the routines, remember to set LOMEM greater than $DD0, the highest location in the alarm routine, so the two won't overwrite each other. The BASIC routine shown here will run as it appears, and will invoke the machine language routine. If you are not bothering with the BASIC, simply JSR $DC9.

After you run the Alarming APPLE and decide to use it for error recovery in your next program, consider these ideas:

Organize the program into error detecting routines, one or more error recovery routines, and an intelligent mainline.

Use an error flag in the recovery routines to inform the mainline.

Use a status flag in the error recovery routines to indicate success or failure of the recovery procedure to the mainline.

Let the mainline make *all* decisions regarding what to do next.

For instance, if you are heavily into structured programming, you might consider a mainline centered on a computed GOSUB with the returns of each routine setting a status number pointing to the next routine. Or you may want to use IFs and GOSUBs tofether in the mainline as each case is decided. The important thing is to route all control decisions — decisions that answer the question: "What next?" — through the mainline. Including error recovery decisions. In fact, *especially* error recovery decisions.

```
  1 REM .    BASIC CALL SEQUENCE
  2 REM .  FOR ALARM PROMPT ROUTINE
  3 REM .
  4 TASK=3000
 10 OFF=0:TASK=200:TRAP=300:ALARM=3529
 95 REM
 96 REM      MAIN LINE SEQUENCE
 97 REM
 98 REM .           -
 99 REM .           -
100 ERR=OFF: GOSUB TASK: IF ERR THEN GOSUB
    TRAP
101 REM .           -
102 REM .           -
110 GOTO 32767
120 REM
121 REM
122 REM
200 INPUT ERR: REM . USE FOR TEST
210 REM
211 REM  PUT ERROR DETECTING TASK HERE
212 REM  REPLACING LINE 200
213 REM
220 RETURN
297 REM
298 REM
299 REM
300 POKE 50,127: PRINT "ERROR";: POKE 50,255
    : PRINT " TYPE A CTRL/C": CALL ALARM
310 REM
320 REM  PUT ERROR RECOVERY ROUTINE HERE
330 REM
340 RETURN
```

Figure 1: Example of a BASIC program invoking the alarm routine in Fig. 2. 3529 is $DC9.

```
0D81-   FF              ???
0D82-   FF              ???
0D83-   AD 30 C0        LDA   $C030
0D86-   88              DEY
0D87-   D0 05           BNE   $0D8E
0D89-   CE 82 0D        DEC   $0D82
0D8C-   F0 09           BEQ   $0D97
0D8E-   CA              DEX
0D8F-   D0 F5           BNE   $0D86
0D91-   AE 81 0D        LDX   $0D81
0D94-   4C 83 0D        JMP   $0D83
0D97-   60              RTS
0D98-   A0 00           LDY   #$00
0D9A-   A2 00           LDX   #$00
0D9C-   A9 47           LDA   #$47
0D9E-   8D 81 0D        STA   $0D81
0DA1-   A9 A0           LDA   #$A0
0DA3-   8D 82 0D        STA   $0D82
0DA6-   20 83 0D        JSR   $0D83
0DA9-   2C 00 C0        BIT   $C000
0DAC-   10 07           BPL   $0DB5
0DAE-   AD 00 C0        LDA   $C000
0DB1-   2C 10 C0        BIT   $C010
0DB4-   60              RTS
0DB5-   A0 00           LDY   #$00
0DB7-   A2 00           LDX   #$00
0DB9-   A9 60           LDA   #$60
0DBB-   8D 81 0D        STA   $0D81
0DBE-   A9 A0           LDA   #$A0
0DC0-   8D 82 0D        STA   $0D82
0DC3-   20 83 0D        JSR   $0D83
0DC6-   4C 98 0D        JMP   $0D98
0DC9-   20 98 0D        JSR   $0D98
0DCC-   C9 83           CMP   #$83
0DCE-   D0 F9           BNE   $0DC9
0DD0-   60              RTS
```

Figure 2: Machine language routine to sound two-tone alarm until ctrl/C is typed. All other input is ignored. To demonstrate, type DC9G to the APPLE II monitor.

6502 Bibliography: Part XIV

William R. Dial
438 Roslyn Avenue
Akron, OH 44320

491. Rainbow 1, Number 4 (April, 1979)

Minch, David "Review: Electronic Index Card", p1.
This program by Bob Bishop emulates a card index file but the reviewer doesn't think it offers substantial advantages.

Editor, "Pascal", p2.
The editor discusses the Pascal Language soon to be available to Apple owners.

Simpson, Rick "Introduction to Assembly Language Programming" pgs. 4-19.
A tutorial article on this important subject.

Watson, Allen "HI-RES Color" pg. 10.
The relationship between the color subcarrier frequency and the color dots; also why there are gaps in vertical and near vertical vectors in colors other than white.

Minch, David "Firmware Review — Programmer's Aid ROM"pgs. 11-12.
The reviewer feels the ROM is a mixed blessing, of limited utility. The Operating Manual is very good.

492. KB Microcomputing No 30 (June, 1979)

Lindsay, Len "PET Pouri" pgs. 6-12.
Discusses a way to protect software, evaluation of cassette quality, new hardware and software, periodicals with PET information, list protection, Trace program, and disabling the STOP key.

Anon.,"Ohio Scientific's Small System Journal"pg.8-11
Discussion of Microcomputer Information Management systems.

Van Dyke, James A. "A Sneaky Interrupt for the 6502" pg. 97.
A novel interrupt using the SO input pin.

493. Creative Computing 5 No 5 (May, 1979)

Kelley, Derek "Beyond the Text Editor"pgs. 32-33.
Program that searches for words, etc., on the Apple.

Hunter, Jim "Peripherals Unlimited Text Editor" p.46.
Upper and Lower case word processing with the Dan Paymar lower case adapter for the Apple and this Text Editor.

Yob, Gregory "Personal Electronic Transactions" pgs. 122-127.
I/O on the PET, Notes on PET Graphics, including higher resolution, a list of PET I/O lines.

494. Personal Computing 3 (June, 1979)

Zimmermann, Mark "Line Renumbering Renumbered" pg.7.
Modifications to the earlier program published in March, for the Apple.

Zimmermann, Mark " 'G' is for Graphics" pgs. 38-41.
An educational picture book for the kids on the PET.

495. Cider Press 2 (May, 1979)

Anon. "Apple Tape Beeps Translated" pg.3.
Why the beeps are on the leader and at the end of programs.

Anon. "May DOM" pg. 2.
The May Disk of the Month has a good mix of programs of different categories; games, utilities, business, etc. Apple.

Hertzfeld, Andy "An Easy Way to Use Text Page 2"pg.3.
Add text to page two graphics using these instructions on the Apple.

Aldrich, Ron "Split Catalog" pg.4.
Split your catalog print-out on the Apple Disk, printing two titles across the page.

Kamins, Scot and Guarriques, Chris "Mini-Word Processor" pg.4.
The Cider Squeezer was specifically designed to aid Apple users to write short articles.

Draper, John T. "Textwriter's Creation" pg.5.
Discussion of how a word processor evolved. Apple.

Espinosa, Chris and Wyman, Paul "CALLS, POKES and PEEKS" pgs. 6-7.
A convenient listing of these important routines for the Apple.

Silverman, Ken "Filling your Memory Cells" pg.7.
An updated listing of the various 16K dynamic RAMs now available together with speed designations, etc. Beware of units of 300 NS or even slower speed.

Nareff, Max J. "Matrix Simulation with the Apple, Part III" pgs. 8-9.
Another installment in this continuing series.

Sullivan, Charles Jr. "Meeting with Sargon" pg.9.
Chess 4 by the author and the wll known Sargo II fight it out on the chess board. Apple.

Anon. "Applibrary" pgs. 10-11.
The Apple Core Library now lists 274 programs in 13 different categories, for the Apple II.

496. Call —Apple 2 No 4 (Apr./May, 1979)

Smith, Ken and Scharen, Rosalie "Analysis of the Great Hello Program", pgs. 4-7.
Analysis of the tricks in this interesting program. Apple.

Golding, Val J. "Integral Data IP 225 Printer: A Review" pgs. 8-11.
How to use this printer with the Apple. A Printer Driver routine by Darrell and Ron Aldrich is given.

Aldrich, Ron "Ron's Page" pgs. 12-17.
HI-Res Screen Dump Routine, Split Catalog Routine, etc.

Suitor, Richard F. "Apple Disk Operating System" pg.17
A discussion including some insights into the details of the new Apple Disk DOS 3.2.

Winston, Alan B. "The Multilingual Apple" pgs 18-19.
Discussion of languages other than Basic for the Apple: PILOT, FORTH, FCL65E, XPLO, Pascal.

Aldrich, Darrell "Monitor Calls, Cross References and Memory Map" pgs 20-23.
Important calls and addresses based on the authors research.

Aldrich, Darrell "The Apple Doctor" pgs 30-31.
How to save a shape table along with an Applesoft Program.

Golding, Val J. "Routine to Center Titles" pg.31.
A simple integer Basic Utility.

497. MICRO No 12 (May, 1979)

Morganstein, David 'Real-Time Games on OSI" pgs 31-33
How real-time games can be written for Challenger systems which use a serial terminal run from the ACIA.

498. Rainbow 1, Issue 5 (May, 1979)

Memory Enterprises "Color-Killer Module" pg.6.
An easy to install module to add the color-killer function to early model Apple IIs.

Watson, Allen III "Another Review of the Programmer's AID #1" pgs. 7-10.
A description of the features of this accessory ROM.

Anon. "An Interview with Phil Roybal of Apple Corp" pgs 11-13.
Phil Roybal, marketing manager for Apple Corp., reveals that coming soon is an "auto-start" monitor ROM that will power up the Apple directly into Basic, or upon hitting reset, also a stop list feature, easier cursor control, etc.

499. Calculator/Computers 3 (Jan./Feb., 1979)

Bobek, Frank "Vats Right" pgs 41-45.
A program to aid physics students studying velocity and acceleration phenomena. For the PET.

500. Calculator/Computers 2 (Nov./Dec., 1979)

Oglesby, Mac "Sinners" pgs 36-38.
Three of Satan's Fiends fight against a group of sinners. For the PET.

501. Dr. Dobb's Journal 4, Issue 6 (June/July, 1979)

Colburn, Don "EXOS-A Software Development Tool Kit for the 6500 Microprocessor Family" pgs. 29-31.
A tool to efficiently and effectively both generate and modify 6500 assembly language programs.

Monsour, Fred J. "Kim Renumber" pg. 47.
A program to renumber KIM-1 Microsoft Basic listings.

502. Stems From Apple 2, Issue 5 (May, 1979)

Armstrong, Kevin "Numeric Sort Routine" pg.2.
A sort program written in Applesoft II.

Reed, Ron "Paddle Speed Control" pg.3.
Control a slow list or program execution with the Apple paddles.

Hoggatt, Ken 'Ken's Korner" pgs 4-5.
A series of tutorials for the Apple user including the use of GET A or GET $A, the use of the mini-assembler, Yes and No statements, etc.

503. Recreational Computing 7, Issue 39 (May/June, 1979)

Feniger, Bob "A Different Way to Float" pg. 6.
A different version of an earlier program for the PET.

Carpenter, Chuck "PILOT for the Apple - An Extended MICRO-PILOT Interpreter" pgs 28-31.
An interpreter in Applesoft.

Saal, Harry "SPOT" pgs. 52-55.
Notes for PET users include information on new models of the PET, a review of Cursor cassette magazine, new books on the PET, and a graphics program listing called CASCADES.

504. Creative Computing 5, No 6 (June, 1979)

Badgett, J. Tom "Strings and Things: Basic String Manipulations" pgs. 86-90.
Tutorial on String variables, etc.

North, Steve "ALF/Apple Music Synthesizer" pgs 102-103
Review of the new Accessory for the Apple which makes possible high-quality computer music.

Soft-One "Apple Graphics Programs" pg. 143.
High Resolution Graphics Utility Set, including character set, lower case, shape vector table assembler, etc.

505. MICRO No 13 (June, 1979)

Putney, Charles B."Harmonic Analysis for the Apple" pgs 5-8.
A program in Applesoft Floating Point BASIC lets the Apple do Fourier Analysis calculations.

Pytlik, William F. "Case of the MIssing Tape Counter" pg.11.
How to locate your files on the PET Cassette.

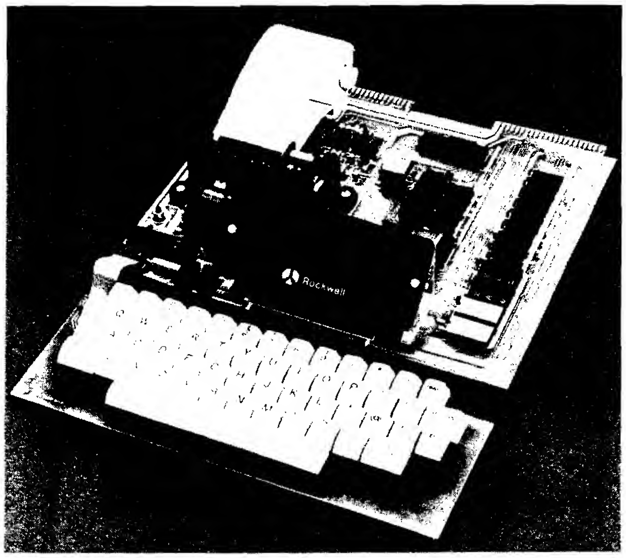
Taylor, William L. "The Basic Morse Keyboard" pgs 13-15.
A Ham program implemented on an OSI system to make an ASCII keyboard act as a "Morse Keyboard."

Rinard, Phillip M. 'A SYM-phony in Stereo" pgs 17-19
This music program uses the 6532 and the 6522's of the SYM to generate stereo music.

Foote, Gary A. 'Sorting with the APPLE II- Part I"pgs. 21-26.
The first installment presents some background material and compares three sorting techniques and gives a program for the SHELL-METZNER sort.

Cass, James L. 'Streamlining the C2-4P" pg.28
Three modifications for the OSI C2-4P to raise its speed, increase cassette through put and add reverse video to the display.

AIM 65 BY ROCKWELL INTERNATIONAL



AIM 65 is fully assembled, tested and warranted. With the addition of a low cost, readily available power supply, it's ready to start working for you.

AIM 65 features on-board thermal printer and alphanumeric display, and a terminal-style keyboard. It has an addressing capability up to 65K bytes, and comes with a user-dedicated 1K or 4K RAM. Two installed 4K ROMS hold a powerful Advanced Interface Monitor program, and three spare sockets are included to expand on-board ROM or PROM up to 20K bytes.

An Application Connector provides for attaching a TTY and one or two audio cassette recorders, and gives external access to the user-dedicated general purpose I/O lines.

Also included as standard are a comprehensive AIM 65 User's Manual, a handy pocket reference card, an R6500 Hardware Manual, an R6500 Programming Manual and an AIM 65 schematic.

AIM 65 is packaged on two compact modules. The circuit module is 12 inches wide and 10 inches long, the keyboard module is 12 inches wide and 4 inches long. They are connected by a detachable cable.

THERMAL PRINTER

Most desired feature on low-cost microcomputer systems . . .
- Wide 20-column printout
- Versatile 5 x 7 dot matrix format
- Complete 64-character ASCII alphanumeric format
- Fast 120 lines per minute
- Quite thermal operation
- Proven reliability

FULL-SIZE ALPHANUMERIC KEYBOARD

Provides compatibility with system terminals . . .
- Standard 54 key, terminal-style layout
- 26 alphabetic characters
- 10 numeric characters
- 22 special characters
- 9 control functions
- 3 user-defined functions

TRUE ALPHANUMERIC DISPLAY

Provides legible and lengthy display . . .
- 20 characters wide
- 16-segment characters
- High contrast monolithic characters
- Complete 64-character ASCII alphanumeric format

PROVEN R6500 MICROCOMPUTER SYSTEM DEVICES

Reliable, high performance NMOS technology . . .
- R6502 Central Processing Unit (CPU), operating at 1 MHz. Has 65K address capability, 13 addressing modes and true index capability. Simple but powerful 56 instructions.
- Read/Write Memory, using R2114 Static RAM devices. Available in 1K byte and 4K byte versions.
- 8K Monitor Program Memory, using R2332 Static ROM devices. Has sockets to accept additional 2332 ROM or 2532 PROM devices, to expand on-board Program memory up to 20K bytes.
- R6532 RAM-Input/Output-Timer (RIOT) combination device. Multipurpose circuit for AIM 65 Monitor functions.
- Two R6522 Versatile Interface Adapter (VIA) devices, which support AIM 65 and user functions. Each VIA has two parallel and one serial 8-bit, bidirectional I/O ports, two 2-bit peripheral handshake control lines and two fully-programmable 16-bit interval timer/event counters.

BUILT-IN EXPANSION CAPABILITY
- 44-Pin Application Connector for peripheral add-ons
- 44-Pin Expansion Connector has full system bus
- Both connectors are KIM-1 compatible

TTY AND AUDIO CASSETTE INTERFACES

Standard interface to low-cost peripherals . . .
- 20 ma. current loop TTY interface
- Interface for two audio cassette recorders
- Two audio cassette formats: ASCII KIM-1 compatible and binary, blocked file assembler compatible

ROM RESIDENT ADVANCED INTERACTIVE MONITOR

Advanced features found only on larger systems . . .
- Monitor-generated prompts
- Single keystroke commands
- Address independent data entry
- Debug aids
- Error messages
- Option and user interface linkage

ADVANCED INTERACTIVE MONITOR COMMANDS
- Major Function Entry
- Instruction Entry and Disassembly
- Display/Alter Registers and Memory
- Manipulate Breakpoints
- Control Instruction/Trace
- Control Peripheral Devices
- Call User-Defined Functions
- Comprehensive Text Editor

LOW COST PLUG-IN ROM OPTIONS
- 4K Assembler—symbolic, two-pass $79.00
- 8K BASIC Interpreter $99.00

POWER SUPPLY SPECIFICATIONS
- +5 VDC ± 5% regulated @ 2.0 amps (max)
- +24 VDC ± 15% unregulated @ 2.5 amps (peak) 0.5 amps average

PRICE: $369.00 (1K RAM)

Plus $4.00 UPS (shipped in U.S. must give ***street*** address), $10 parcel post to APO's, FPO's, Alaska, Hawaii, Canada, $25 air mail to all other countries

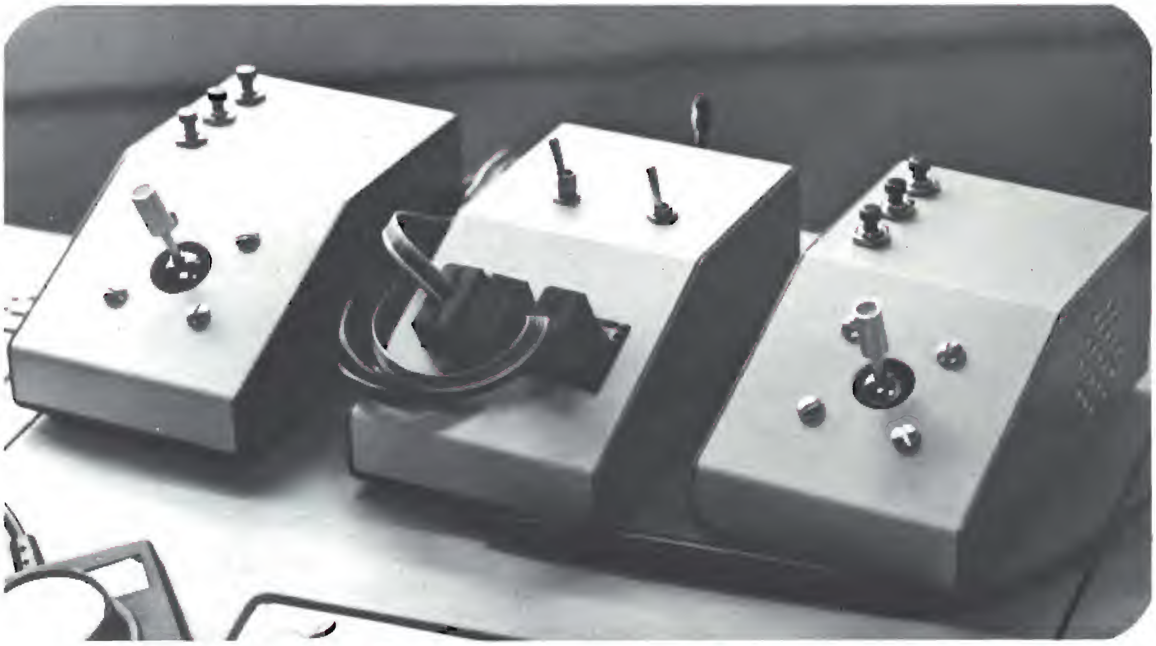
We manufacture a complete line of high quality expansion boards. Use reader service card to be added to our mailing list, or U.S. residents send $1.00 (International send $3.00 U.S.) for airmail delivery of our complete catalog.

RNB ENTERPRISES INCORPORATED

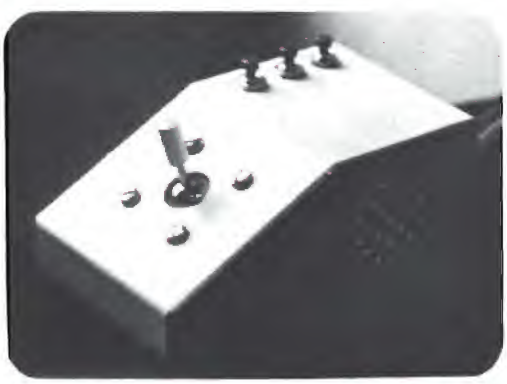
2967 W. Fairmount Avenue
Phoenix AZ 85017
(602)265-7564

master charge VISA

APPLE II® JOYSTICK & EXPANDA·PORT



EVERY APPLE II OWNER SHOULD HAVE ONE!



JOYSTICK $49.95

The PROGRAMMA JOYSTICK is an input peripheral that attaches to the APPLE II Computer's game I/O Port. The JOYSTICK is a must for the serious game player, and it offers a degree of linearity not currently available with other joysticks. The ease of manueverability and the availability of the "functional" switches make the PROGRAMMA JOYSTICK a much needed enhancement to any APPLE II Computer System owner. The PROGRAMMA JOYSTICK comes completely assembled and tested, including a User's Guide.

EXPANDA-PORT $49.95

The PROGRAMMA EXPANDA-PORT is a multi-port expander for the game I/O port of any APPLE II Computer System. In addition to allowing expansion for up to six devices, the EXPANDA-PORT contains a built-in speaker that replaces the function of the Apple II's speaker. The switches on the EXPANDA-PORT allow for the selection of the specific device desired and for the switching of that device. No unplugging of any device connected to the EXPANDA-PORT is required. The PROGRAMMA EXPANDA PORT comes completely assembled and tested, including a User's guide.

The PROGRAMMA JOYSTICK and EXPANDA-PORT are available on a limited basis through your local computer dealer. Apple II is a registered trademark of Apple Computers, Inc.

PROGRAMMA INTERNATIONAL, INC.
3400 Wilshire Blvd.
Los Angeles, CA 90010 (213) 384-0579 • 384-1116 • 384-1117

PROGRAMMA

Computer System Products